Implementation of Projects for Periodic Task Execution

September 2021

AUTHOR:
Rodrigo Bermúdez Schettino
Technische Universität Berlin, Germany
CERN openlab Summer Student 2021

SUPERVISOR:
Dr. Ulrich Schwickerath
CERN Section IT-CM-LCS

CERN openlab

PROJECT SPECIFICATION

The authenticated cron (acron) service enables users to schedule regular task execution. This is achieved by creating crontab entries for Kerberos-authenticated cron jobs (Acron, 2021). In other words, it provides a system to centrally schedule user cron jobs that will be executed on a user-defined target host and interval; such jobs are called acron jobs.

A highly requested feature by acron users has been the ability to share jobs between multiple users. This is especially useful when acron is used for services managed by teams rather than individual users.

The goal of this project is to implement acron projects to allow several users to manage a common set of acron jobs in a secure manner—i.e., without the need to share any secrets for the user who is running those jobs. A second objective is to implement it in a reusable way—i.e., without CERN-specific code; the source code of the service should be released to the open source.

ABSTRACT

On Unix systems (like Linux), the cron daemon allows users to periodically execute specific tasks. The acron service at CERN extends this concept and provides a way to execute such tasks in a centralized and secure way and allows scheduling tasks with Kerberos and AFS credentials on central services, like lxplus, at CERN. The acron service is widely used across CERN, especially—but not exclusively—by scientists from the experiments, to automate recurring task execution. The scheduled tasks which are automatically executed are called acron jobs. This service helps some of our users to collect, pre-process, and analyze collision data from the experiments. It has been in service since 1996 (Toebbicke, 1996) and has undergone few changes over the years. However, in the past two years a re-design phase has been put in motion.

The aim of this project is to design and develop a secure way to share acron jobs between multiple users called acron projects and, thus, mitigate this shortcoming of the acron service. The present work builds upon an existing draft dating back to 2019. Such draft was revamped to meet the updated expectations of the service.

In conclusion, our contribution is the secure implementation of job sharing for periodic task execution—i.e., acron—; we restrict its availability to service accounts to eliminate the impersonation risk, this feature is only available in the deployment at CERN because service accounts are CERN-specific. After careful analysis, CERN Computer Security approves the design of this novel feature. Finally, a risk assessment and code review of the entire acron service is carried out by Computer Security.

ACKNOWLEDGEMENTS

First and foremost, I am profoundly grateful to my supervisor, Dr. Ulrich Schwickerath, whose expertise was invaluable in sharpening my thinking and bringing my work to a higher level. Ulrich’s drive has been a constant source of motivation. I especially appreciate his steady and structured support before and during the project. I wish to extend my special thanks to Zhechka Toteva for supervising my work when needed.

I thank CERN organizers for making this year’s programme possible despite the obvious difficulties. The lectures were particularly interesting because they sparked my curiosity in new fields of research and allowed me to deepen my knowledge in other topics. I would like to acknowledge the IT-CM-LCS section members for fostering a productive and enjoyable work environment.

On the personal side, I thank my family for their unconditional and continuous support. I dedicate this milestone to them.

TABLE OF CONTENTS

1. INTRODUCTION

Regular task execution lies at the core of maintaining an organization’s infrastructure in an efficient manner. The candidate for this use case on Unix-like operating systems is usually the well-established software utility cron. However, the execution of cron jobs is limited to a single machine. At the same time, from a Systems Reliability Engineering (SRE) standpoint, this defines also cron’s failure domain—single machine (Google, 2015). At CERN, we are interested in running services at scale. Consequently, the need of scaling this service to multi-machine execution. A further requirement is multi-factor authentication (MFA) support.

In many cases, we need to perform jobs on a recurring basis. A first example is the interactive login service at CERN: In this use case, short-lived nodes are provisioned in the lxplus cluster to authenticate users. These nodes are required to perform server monitoring tasks (e.g., health checks) regularly, which require authentication. A second example is data processing for CERN experiments: This ranges from collection of data to its analysis.

The further sections of this report are structured as follows: In the Background section, the legacy acrontab service and its successor are introduced. In the Acron Projects section, the summer project is described. Finally, in the Conclusion section, we present an outlook and propose a possible future summer project on this topic, as well as a review of the status of the service.

2. RELATED WORK

The acron service—also spelled as ACRON or Acron—stands for authenticated cron. Additionally, you may also encounter the term acrontab. In this section, the different versions of the service will be introduced, and historical background will be briefly provided.

a. LEGACY ACRON

Originally, acron stood for AFS cron (Toebbicke, 1996). The service is accessed via a command-line tool (CLI) tool called acrontab. The name is derived both from the Linux command crontab to manage cron (Linux) entries and the crontab file, which contains entries in the format: schedule followed by the command. The name crond also refers to the daemon to execute scheduled commands. Precisely the usage of cron as a backend introduces a single point of failure (SPOF). Additionally, although the service benefits from a hot standby server, it must be switched manually.

The code is written in perl and C and is integrated with Andrew File System (AFS) libraries. Moreover, it uses a home-grown protocol called Authenticated Remote Control (ARC). These were the main motivations to re-design the service.

b. ACRONNEXT

Acronnext is the re-implementation of acron. Unlike the legacy acron, it only requires sshd to be installed on client and a password-less authentication method, e.g., Kerberos credentials. The desiderata for this novel service are defined as follows: high availability, scalability, security, and ease of use (Ganz & Schwickerath, Future of the acron service, 2019). The service meets these criteria: high availability via automatic failover, scalability via load balancer, security via credentials management using Kerberos, and ease of use by offering compatibility with cron-like syntax (Ganz & Schwickerath, Status update for the acron service, 2020). See Figure 1 for a diagram of the architecture of acron.

The software leverages modern tooling. Frontend and backend are written in python3 and the interface to the backend is via a REST API. The connection to the target machines, where jobs are executed, is done using SSH. Acron is installed at CERN on CentOS 7 and CentOS 8 hosts. Therefore, the installation of acron for these operating systems is done using RPMs.

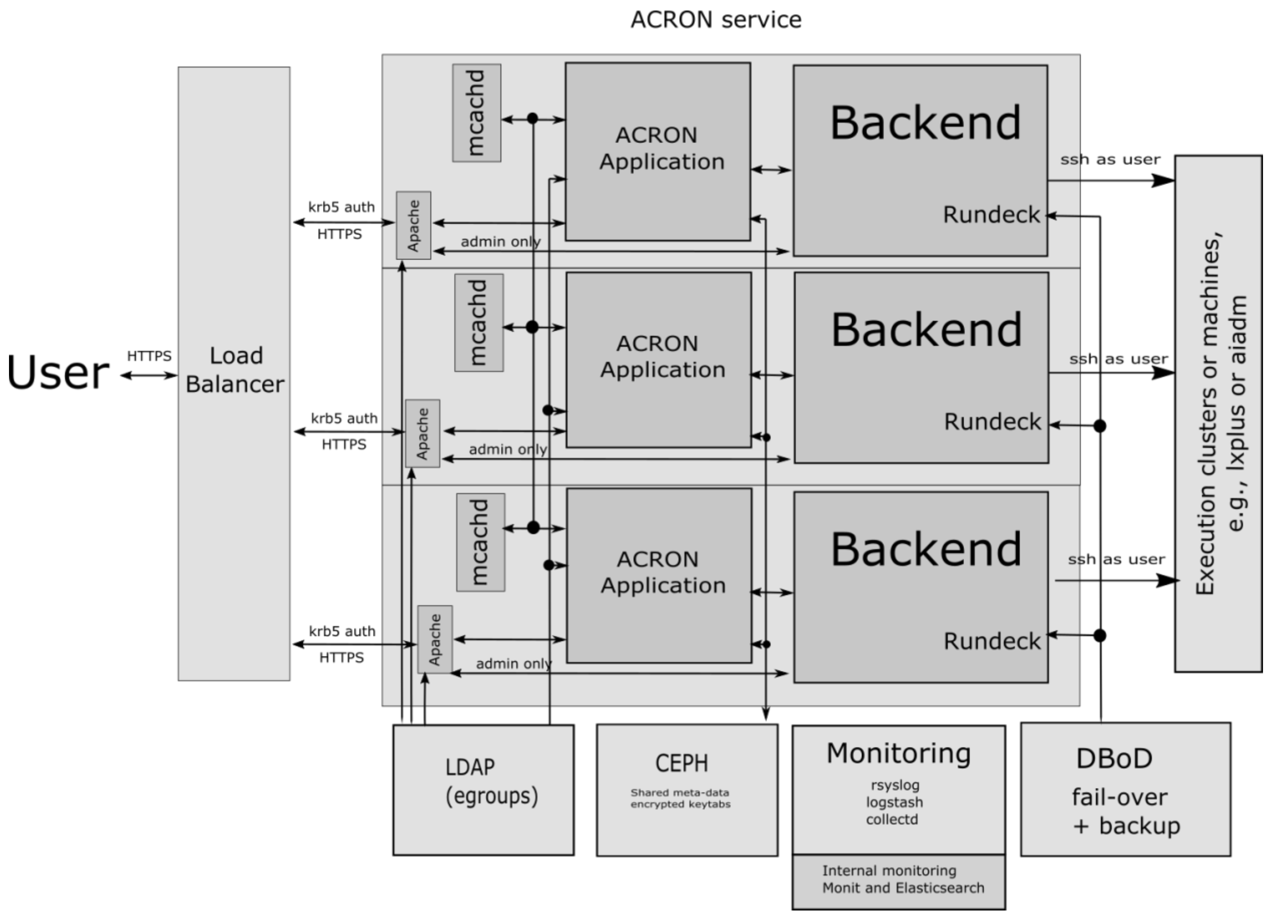


Figure 1: Architecture of acron service by Dr. Ulrich Schwickerath

c. IMPORTANCE OF THE ACRON SERVICE

At CERN, the acron service is used to run workflows related to the experiments. It is suitable for any short-lived periodic workload which requires strong authentication—e.g., to access storage systems or other services. The service has significant user base distributed over the legacy and the current versions of service. In Figure 3, the number of distinct users per day is displayed. Many of these users execute jobs on a regular basis, see Figure 2: Number of jobs per day.

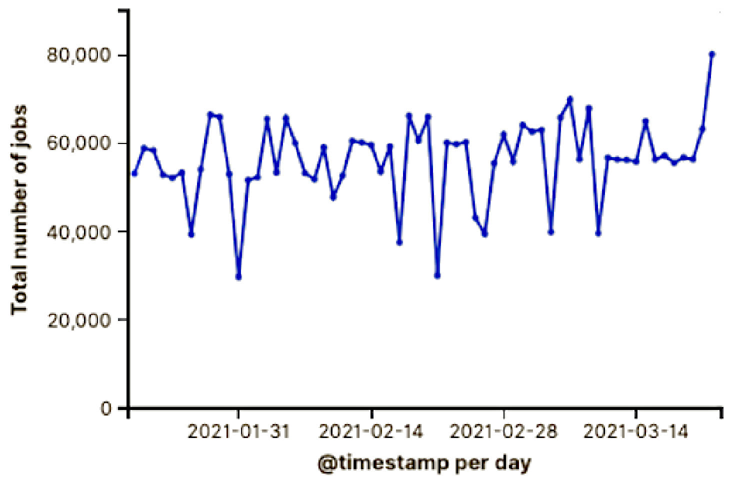


Figure 2: Number of jobs per day

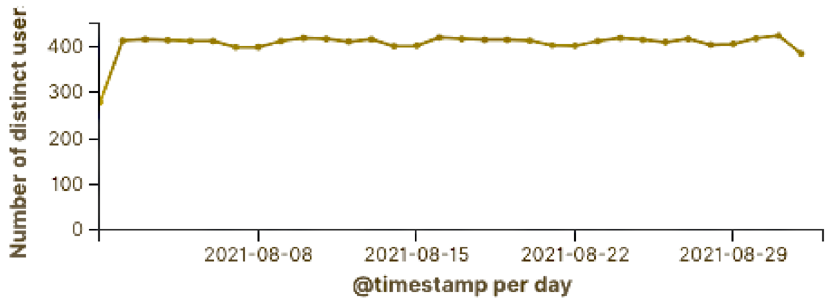


Figure 3: Number of distinct users per day

3. ACRON PROJECTS

Often, acron jobs perform tasks across services—e.g., to check availability of other services—, which are run by groups of people rather than individuals. In this case, several people need to be able to manage such jobs (Schwickerath, 2021).

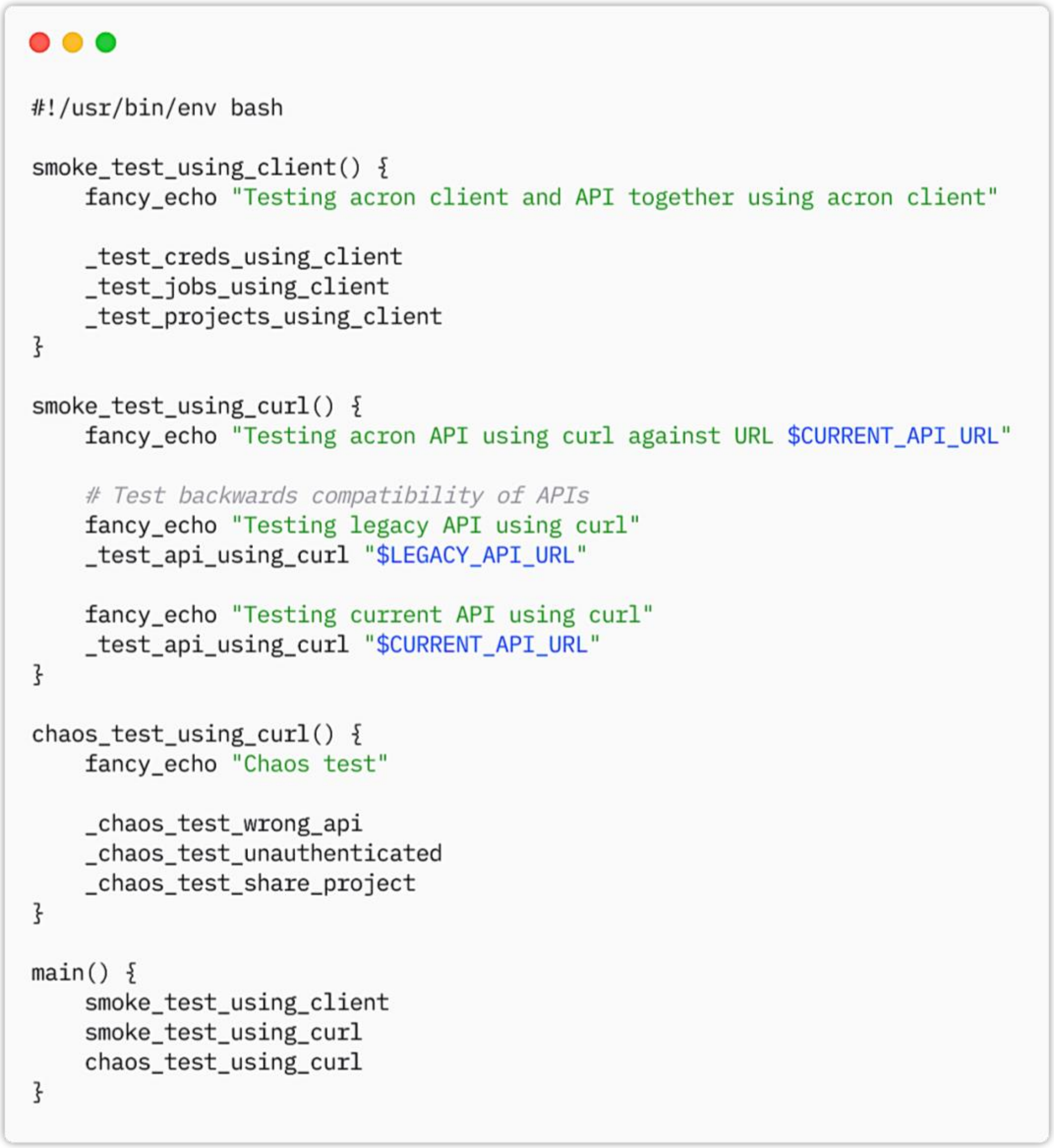
The implementation of acron projects constitutes the main objective of this Summer Student project. It should give users the ability to share acron jobs securely. It requires the revision of an existing draft for acron projects dating back to 2019.

In the backend, the scheduler is called by the API to process user requests and it oversees the execution of jobs. The scheduler is defined as an abstract class written in python to allow for interchangeability of the implementation. We consider three alternatives for the implementation of the scheduler: Nomad, Cron, and Rundeck. The current implementation of the scheduler uses Rundeck.

a. TESTS

Before tackling acron projects, a testing suite was implemented to prevent software regression. Initially, the test had two main focal points: credentials and jobs. These illustrate acron's top-level subcommands in the client. Eventually, projects were added to the testing suite after its implementation.

The testing suite comprises two smoke tests and a chaos test. The smoke test is conducted both with the official acron client and curl; curl sends HTTPS requests directly to the API. Additionally, we leverage a Chaos Engineering approach to evaluate our system under stress and validate its response. In summary, we run three steps: smoke test with client, smoke test with curl, and chaos test using curl. The chaos test ranges from sending requests to invalid URLs to interacting with the service as an unauthenticated user. The test suite is written in bash and a snippet is shown in Figure 4. Disclaimer: The snippet in the figure should serve as a glimpse into the architecture of the test suite; a significant portion of the code was omitted.

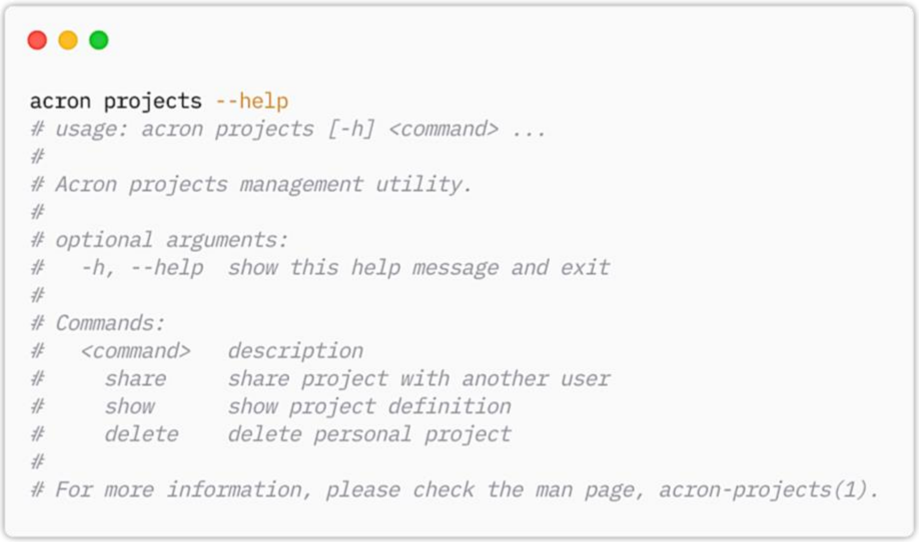


```
#!/usr/bin/env bash

smoke_test_using_client() {
    fancy_echo "Testing acron client and API together using acron client"

    _test_creds_using_client
    _test_jobs_using_client
    _test_projects_using_client
}

smoke_test_using_curl() {
    fancy_echo "Testing acron API using curl against URL $CURRENT_API_URL"

    # Test backwards compatibility of APIs
    fancy_echo "Testing legacy API using curl"
    _test_api_using_curl "$LEGACY_API_URL"

    fancy_echo "Testing current API using curl"
    _test_api_using_curl "$CURRENT_API_URL"
}

chaos_test_using_curl() {
    fancy_echo "Chaos test"

    _chaos_test_wrong_api
    _chaos_test_unauthenticated
    _chaos_test_share_project
}

main() {
    smoke_test_using_client
    smoke_test_using_curl
    chaos_test_using_curl
}
```

Figure 4: Snippet from the test suite written in bash

b. IMPLEMENTATION OF PROJECTS

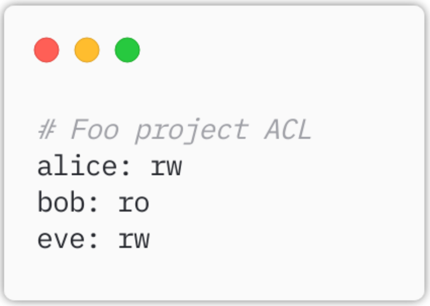
By design, there is a one-to-one relationship between users and projects—i.e., each user has only one project. Project owners can grant other users read-write or read-only permissions to their project; abbreviated as *rw* and *ro*, respectively. A key-value database is stored in a file for each project with the format user as key and permissions as value—see Figure 6. These files are shared between servers to maintain consistency.



```
acron projects --help
# usage: acron projects [-h] <command> ...
#
# Acron projects management utility.
#
# optional arguments:
#   -h, --help   show this help message and exit
#
# Commands:
#   <command>    description
#     share      share project with another user
#     show       show project definition
#     delete     delete personal project
#
# For more information, please check the man page, acron-projects(1).
```

Figure 5: Usage of acron projects in CLI

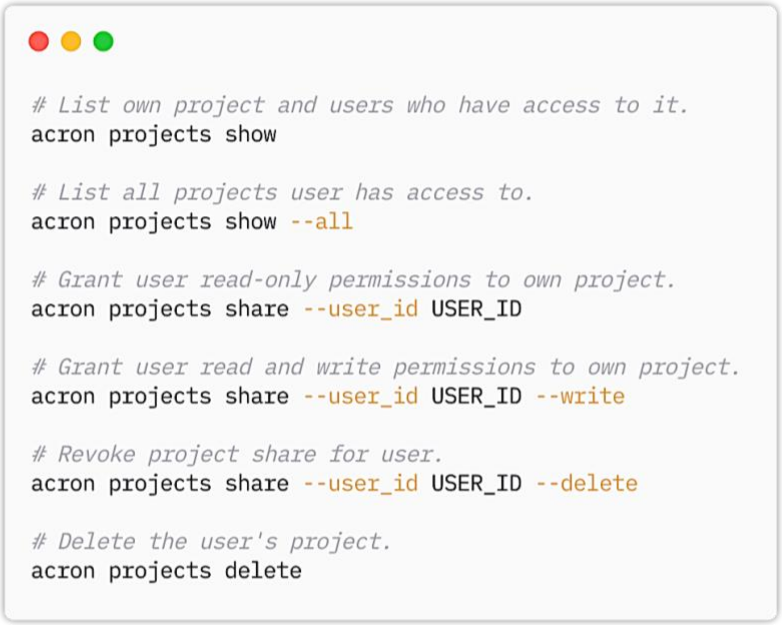
Jobs in a project are executed with the credentials of the project owner, regardless of who created or edited the job. Jobs have full access to services and data of the executing user, specifically, to the owner's home directory and files on the executing machine (Pearce, 2016), regardless of whether the home directory is locally available on the executing machine, or if it is placed on the Andrew File System (AFS). An analysis of potential risks and implications is conducted in section c.



```
# Foo project ACL
alice: rw
bob: ro
eve: rw
```

Figure 6: Example file contents of project ACL

Projects was integrated as a subcommand into the existing acron CLI command. The documentation is available to the user as help text—shown in Figure 5—and in a man page, also mentioned in this help text. The man page provides usage examples—see Figure 7—. Note that user confirmation is required for destructive actions to prevent accidentally deleting a project, for instance. Additionally, to secure the application, user input is validated both client- and server-side to prevent code injection.



```
# List own project and users who have access to it.
acron projects show

# List all projects user has access to.
acron projects show --all

# Grant user read-only permissions to own project.
acron projects share --user_id USER_ID

# Grant user read and write permissions to own project.
acron projects share --user_id USER_ID --write

# Revoke project share for user.
acron projects share --user_id USER_ID --delete

# Delete the user's project.
acron projects delete
```

Figure 7: Example usage of acron projects in CLI

c. CYBERSECURITY ANALYSIS

The acron service has audit logging in place to track the change history. However, one of the major risks we identified is outlined in the following use case.

i. Setup

User O is project Owner. User S is one of the users with whom the project was shared with read-write permissions (can view and edit jobs). Job J is a specific Job in user O's project.

ii. Scenario

User O has jobs running on a target node. These jobs, by design, run with user O's credentials. User O's project is shared with user S. User S edits job J to execute malicious commands*.

* Jobs in a project can access user O's AFS and run in O's name (this could lead to impersonation). Furthermore, user S could authenticate as user O to third-party services supporting Kerberos credentials.

iii. Forensics

We would be able to find out that user S edited the job and which changes were made. Nevertheless, the job would have run in user O's name, which is delicate, to say the least.

iv. Discussion and Mitigation

Using the credentials of the user who last updated the job could lead to failure of jobs and would imply a radical change in the codebase. We advise users to create a dedicated account for shared acron jobs only, which can be achieved at CERN using the concept of service accounts.

CERN Computer Security conducted a Threat & Risk Assessment and a Code Review of the acron service after the implementation of projects. After thorough these reviews, the design and implementation of projects is approved, including the limitation of acron projects to service accounts.

4. CONCLUSION

During this summer project, user and developer documentation was improved, client- and server-side bugs were fixed, overall software robustness was enhanced using the test suite, and new features were implemented. Selected examples of such features are: acron projects, support for custom job names, and backward compatibility of API versions. The implementation of acron projects was accomplished four weeks ahead of planned.

After approval from Computer Security for the implementation of the acron service, the source code was released to the open source on GitHub.

5. REFERENCES

Toebbicke, R. (1996, June 15). *Scheduling AFS Cron Jobs -- The acrontab Command*. Retrieved from CERN Document Server: http://cds.cern.ch/record/1017936

Acron. (2021). *Acron service mandate*. Retrieved from Acron documentation: https://acrondocs.web.cern.ch/mandate/mandate/

Pearce, A. (2016, June 21). *Running periodic Kerberos-authenticated jobs with acron*. Retrieved from https://alexpearce.me/2016/06/running-kerberos-jobs-with-acron/

Ganz, P., & Schwickerath, U. (2019, May 23). *Future of the acron service*. Retrieved from https://indico.cern.ch/event/814818/contributions/3425434/attachments/1849752/3036243/acron.pdf

Ganz, P., & Schwickerath, U. (2020, February 13). *Status update for the acron service*. Retrieved from https://indico.cern.ch/event/884283/contributions/3726173/attachments/1987120/3311515/acron.pdf

Schwickerath, U. (2021, April 8). *ACRON next generation: Status and deployment plans*. Retrieved from https://indico.cern.ch/event/1025180/contributions/4304358/attachments/2222289/3763441/08042021.pdf

Linux. (n.d.). *cron(8) - Linux man page*. Retrieved from https://man7.org/linux/man-pages/man5/crontab.5.html

Google. (2015, March). *Distributed Periodic Scheduling with Cron*. Retrieved from https://sre.google/sre-book/distributed-periodic-scheduling/

A. LIST OF ACRONYMS

ACRON Authenticated Cron

AFS Andrew File System

API Application Programming Interface

ARC Authenticated Remote Control

CLI Command-Line Interface

IT-CM-LCS IT Computer & Monitoring, Linux, and Configuration Support Section

LB Load balancer

MFA Multi-Factor Authentication

OTP One-Time Password

REST Representational state transfer

RPM RPM Package Manager

SRE Systems Reliability Engineering

SSH Secure Shell