Privacy Request Management System

An architectural proposal for an in-house privacy management system at Vox Media.

Introduction

We are already building the pieces of this system, between consent collection forms, Google docs, JIRA workflows and deletion queries. A unified pipeline will allow us to:

- Automate the deletion tasks and data requests
- Manage audit records without creating *additional* user data

Stakeholders:

- Greg Tyree
- Patrick Hogan
- Marie Connelly
- Brandon Morrison
- Livia Labate
- Jory Ruscio

Principles

- **Keep it simple**: Let's not over-engineer this - it's a small collection of components ultimately intended to do one thing: manage user privacy requests
- **Keep it isolated**: User data that comes into this system as part of the privacy management/deletion process shouldn't go back out again
- **Keep it scalable**: Legislation and expectations are changing rapidly, so whatever system we build needs to be flexible to account for an uncertain future.

Responsibilities

There are four immediate groups of responsibilities:

1. **Collect the request from the user** – API that can accept user privacy requests from a dedicated Chorus contact form, or via email. Privacy requests currently fall into two categories: access requests (records, or “what information do you have about me”) and deletion requests (deleting data such as comments or content in Chorus/Clay)
2. **Automated action** – Execute access/deletion tasks
3. **State Management dashboard** – Community team can approve and see the state of requests and intervene if necessary
4. **Notify the user** - When all of the privacy request steps have been completed, close the request and email the user (with confirmation of a deletion, or the results of an access request)

Application Flow

A request comes in from the user, through either a contact form or an email message. The request payload is posted to an API endpoint.

The initial request is stored; an entry is created that appears on a dashboard. At that point, the dashboard indicates only that the request has been received and that the confirmation process has begun.

After the confirmation is completed, the dashboard entry shows a list of deletion/access tasks that should be executed (based on the content of the request).

That list of tasks should be approved by a human, who will have the option of adding or removing tasks from the stack. Once approved, the deletion/access tasks will begin to execute.

The dashboard shows tasks in the following states:

- inactive/unstarted
- in progress
- completed successfully
- failed (a failed state should include a specific error message)

As each task is completed, the dashboard updates to show its change in status.

When all the tasks are completed, the request is marked as closed, and an email notification is sent to the user.

Per our legal counsel (Patrick Hogan), any user data created in order to satisfy a privacy request can be retained, so there is no reason to delete request data at the end of the process.

Components Detail

The three main components - API, status management dashboard, and task scheduler - will stay decoupled, running as separate small apps but living in the same repo for now.

API

Accepts new deletion/access data requests (via form submission)

- Email request responds by redirecting user to the contact form
- Form submission triggers a confirmation cycle which leads into the task flow

Handles an automated confirmation flow (to determine what the confirmation flow should look like and what elements must be included, we can refer to the docs 'Data privacy request workflow' and 'Community | Canned responses')

Parses and saves the request to an isolated store (a dedicated PostgreSQL or MySQL instance). This process should be able to evaluate the content of the request and determine which deletion/access tasks are required in the next step. In the event that those tasks can't be determined, a human will be able to select the tasks manually through the dashboard.

The initial user request payload will probably not include enough data to allow us to identify records for some of the deletion/access tasks. The API will need to make a few additional requests to assemble things like, for example, Chorus username or membership in Chorus communities. The full list of values currently being assembled for manual processing in Jira is available in the GDPR/CCPA records request or data deletion template.

Dashboard

For login/secure access, we'll use Okta. Per Jennifer Kramer: "It assumes that everyone who ever uses it will be an actual honest-to-goodness Vox Media Employee. Reach out to the IT folks to figure out how to get going on this. The SRE team has integrated okta into several apps we host ourselves (grafana, spinnaker) so they can probably help once you get to implementation."

The start page will feature a list of active requests (a request will appear here as soon as soon as it is received by the API; unconfirmed requests will drop off after 7 days)

Each request has its own page that displays:

- A minimal amount of user information (name? user id?)

- Date the request was received (alternately, a deadline date by which the request needs to be completed)
- A checklist of tasks that have been determined for the user. The API will produce an initial list based on the user request.
    - A human should then be able to edit that list from a dropdown or some other collection of all available tasks.
- A button to approve the list of tasks and begin running them
- An indication of the status of each task (inactive/running/success/fail), along with a space for an error message if a task fails

An admin section for managing:

- Confirmation messaging (canned messages will be stored in the database, and community users should be able to add and edit them here)
- Activate/deactivate available tasks (tasks could just be made available as soon as they're added to the code repository, but adding an active/inactive option provides an extra level of control; we might also want to store metadata about the tasks themselves and manage it here)

Members of the Community team will be able to interact with the dashboard to:

- Add/approve tasks on a specific request
- Comment on a specific request
- Approve/start the task queue

Task Scheduler

Automated deletion/access tasks:

- Access pathways ("what do you know about me?")**
- Chorus account deletion (Brandon Morrison, Livia Labate, Curtis Schiewek, Jamie McCarthy, Audience team)
- Chorus contributor content (comments) (Brandon Morrison, Livia Labate, Curtis Schiewek, Jamie McCarthy, Audience team)
- Coral (Kim Gardner)
- Campaign Monitor/newsletter user data (Barbara Shaurette, Emma Merlis)
- Revue/newsletter user data (Emma Merlis)
- SailThru/newsletter user data (TK, Brooklyn Presta)
- NYM data stores? (Prratek Ramchandani, Joy Yu, Jeff Pope)
- Clay (Jeff Pope)
- Google Analytics User Reports user deletion (Barbara Shaurette)
- BigQuery (Google session data) deletion (Joy Yu)

The above list is only intended to be a sampling of the types of tasks that will eventually be integrated into this system. A more complete list of all the records request/deletion tasks

currently being executed can be found in Jira, in the GDPR/CCPA records request/data deletion template, as well as in the Data privacy request workflow master document.

Task execution is triggered manually via the dashboard - The full task stack should be approved and triggered by a Community team member, but we'll also need to provide the ability to retry failed tasks or run individual tasks outside of the stack when necessary.

Task runner overall structure:

- Task queue (I would like to try out Redis Queue for this, but it should be pointed out that this app will probably never be called on to manage a large volume of requests, so building a queuing system might be over-engineering.)
- We will need to track the state of each task - `inactive` (or `unstarted`), `running`, `success`, `failed`
- We will need to impose some kind of ordering/scheduling on the tasks.
    - Chorus and Coral deletions can be processed as they're received, but given that some of the jobs can impact Chorus performance, we should schedule them and build in notifications to the Chorus team before they are run.
    - BigQuery deletions should be added to a queue and batched for processing on a regular basis (stored to a `batch` table, flagged as `inactive` until a weekly/monthly job - the timeframe is TBD - picks them up from the table and executes them)
    - Some tasks might be related to areas where the user comes back and generates more data before the deletion has been completed - those tasks should be executed last.

Additional details:

- This will be a Python/Javascript project:
    - Flask or FastAPI for the API
    - React.js for the dashboard
    - Flask & other libraries for the task scheduler/backend
- Any deletion processes with dependencies on one another should be contained in a single task.
- Error handling: Every task should return a clear success or failure status, and a failed task should include a clear error message that can be bubbled up to the dashboard.

Closing the request

- An automated process, this could just be treated as one of the task modules, albeit one that always runs as the very last task.
- This task will include the logic that marks the request as `closed` or `completed` and triggers a notification to the user (notification language is outlined here)
- For access requests, the notification to the user should just include general information about the kinds of data we have stored without returning specific data. The notification should offer the user the option to delete (an affirmative reply kicks off a new request).

Open Questions:

- Credential management: Since this system will require us to connect with all of our data stores, we'll need to securely manage credentials to read/write to many systems. (We use Chef data bags for most of our credential storage now, but I'll need to check in with SRE about what the preferred platform is these days.)
- API: What information will we need to require in incoming requests? What is the bare minimum we need to be able to find user data and process deletions downstream?
- Deletion/access tasks:
    - ** Who is responsible for the existing access tasks?
    - What will be required to configure connections between this app and the relevant data stores on both the Vox Media and NYM sides?
    - What request data is going to be required by each task? What needed request data will there be in common?

Reference Links

The current manual process is outlined here: Data privacy request workflow
Responses (used in the initial confirmation cycle): Community | Canned responses
JIRA board: https://vmproduct.atlassian.net/jira/software/projects/DPR/boards/176

CCPA: https://oag.ca.gov/privacy/ccpa
GDPR: https://gdpr.eu/checklist/

Incoming request data:

- The Chorus contact form: https://github.com/voxmedia/sbn/pull/9863/files (contacts: Liv & Curtis, Brandon)
- Example request data from Zendesk: https://voxmedia.slack.com/archives/CPJDM3CCU/p1597934886002800
- Additional browser storage values: https://app.slack.com/client/T024F4436/CPJDM3CCU
- NY Media Contact Form (Jeffrey Pope, Greg Tyree)

Tasks:

- Campaign Monitor (for user deletion and to get user details): https://www.campaignmonitor.com/api/subscribers/
- Existing deletion scripts (GA, BQ, etc.)
    - Google Analytics User Reports user deletion (Barbara Shaurette)
    - Chorus rake tasks

Presentation:
https://docs.google.com/presentation/d/1_v90ACoS0I_qsJvkKkdw4wdRYFRDDsfCxSLYvc5AvxQ/edit?usp=sharing
Recording:
https://voxmedia.zoom.us/rec/share/vNwrMO7LxDxOGYnCuF7AcbwFH4m5X6a82nUW_aANmE1zgOWZgoTURh6vt8GwuhJT?startTime=1598983485000

Planning

Technical Analysis (draft)
Project Planning Doc