



Implementation Proposal

[Implementation proposal](#)

[Glossary](#)

[Functionality](#)

[Services](#)

[Sequence diagram](#)

[Battle plan creation](#)

[Draft of an example plan](#)

[Fuzzy preamble from notes \(outdated\)](#)

[Scraping](#)

[Email formulation + sending](#)

[User-facing application/dashboard](#)

[“You have been pwned” pages](#)

[Profile data](#)

[Loot data](#)

Implementation proposal

Glossary

Functionality

Note: this is high-level functionality needed for the product, a concrete proposal is in “Services”

Profile Data

Scraping → scrapes data and saves it in the profiles schema for production-ready use

Third-party service ingestion

Manual data entry

Dashboard: Login, view profile schema data, monitor, and—at least initially—create campaigns.

Attack Plan formulation: An attack could be started manually by a user from the dashboard, or automatically given some event (e.g. periodic check on the db state, etc.), the "plan formulation" part involves taking a bunch of decisions and parameterizing the attack for the executor (e.g. the logic in charge of sending the email).

Attack Plan execution (Phishing): sending emails, serving the phishing websites, registering inputs submitted by the user, checking for email replying to our fishing attacks, etc. Responsible for starting the attack, keeping track of it and storing/serving outcomes.

Services

Concrete applications which cover the items in the **functionality section**.

Profile data:

- scraping
- exposing an admin dashboard for manual data input by us
- ingestion from third parties
- exposes said data through API endpoints. I have a hunch that said read-only endpoints would be better served through graphql rather than the usual REST stuff but I might be mistaken. Anyway, let's start simple
- later: an API that allows users to clean-up/modify their own data? (we can already do that through the admin panel) Has the advantage of having a single point of decision when it comes to constraints about what's going in (imo

preferable to having multiple writers of the profiling data writing directly to the DB)

- python application, django + a celery worker for queuing the scrape tasks, wouldn't mind not having a celery worker and just going with an application-level scheduler for the time being, I kinda need to freshen up on what's the current state of the art when mixing these (e.g., scrapy?)

Dashboard:

- user-facing dashboard
- works at the campaign level
- request attacks to the attack service, each campaign can include N attacks
- Shows campaigns and their details (e.g. attacks, breaches).
- typescript application

Attacks

- exposes API endpoints to POST attack requests, based on target and the available profile data it will asynchronously carry out the attack. Example
 - an attack request is submitted by POST
 - you get an attack payload back with uuid, etc.
 - you can use the uuid to retrieve the status of the attack, (loot data?), etc.
- exposes phishing pages (like forms to submit your psw), checks email inboxes for loot data, etc.
- keep track of all attack interactions
- exposes API endpoints to read the status and attack interactions of an attack
- django application, might need a django worker later, but let's start with an internal scheduler

Sequence diagram

```
sequenceDiagram
    participant User
    participant Dashboard
    participant AttackerService
    participant ProfileDataService


    User->>Dashboard: Create account/team
    Dashboard -->>Dashboard: Create namespace for team
    User->>Dashboard: Start campaign
    Dashboard->>AttackerService: Attack 1, 2, 3, ...N
    Note over Dashboard,AttackerService: POST payload(namespace, company info, target info)
    AttackerService-->>Dashboard:
    AttackerService->>ProfileDataService: Check if there is enough data to attack
    Note over AttackerService,ProfileDataService: GET payload(namespace, ...)
    ProfileDataService->>ProfileDataService: Upsert namespace if not exists
    ProfileDataService->>ProfileDataService: Start scraping data if lacking
    AttackerService->>ProfileDataService: Check if there is enough data to attack
    AttackerService->>ProfileDataService: Check if there is enough data to attack
    ProfileDataService-->>AttackerService:
    AttackerService->>AttackerService: Attack 1, 2, 3, ...N are performed
    User->>Dashboard: Check dashboard
    Dashboard->>AttackerService: Get attack status data
    AttackerService-->>Dashboard:
    Dashboard-->>User: Results
```

Battle plan creation

How does a good battle plan look? Things to consider ...

- Granularity—detail of the plan:
 - Does it contain the exact wording that the agent should use for phishing? Or does the agent have more agency?
- Temporality—when the plan is executed:
 - Does the plan (or steps of the plan) include timing information? E.g. send an email at 12:00, then send a follow-up SMS message two minutes later.
- Goals—what the plan aims to achieve:
 - How many types of plans do we want to support?

Draft of an example plan

In this example, we're sending a phishing email followed up by an urgent SMS reminder. For more detail about what can be included in the battle plan, see the [Domain model](#) document. 

- The `"steps"` are the execution plan
- The `"context"` is data that is accessible in the prompt templates

```
{
  "campaign_id": "59bc0fcc-2f2b-4143-a68f-4004bfa006b2",
  "goal": "click_phishing_link",
  "delivery": "email",
  "steps": [
    {
      "name": "Send initial email",
      "action": "send_phishing_link_email",
      "at": "2023-03-21T12:42:23.001Z",
      "prompt_template_id": "ed04b341-f007-4bd5-9542-3895023183bc",
    },
    {
      "name": "Urgent SMS follow-up",
      "action": "send_sms",
      "delay": "5m",
      "prompt_template_id": "d14beed5-bfd1-4a4c-b157-ed6225e535bc",
    }
  ],
  "context": {
    "phishing_link": "https://dont-click.me/59bc0fcc-2f2b-4143-a68f-4004bfa006b2",
    "recipient": {
      "called_name": "Jimmy",
      "first_name": "Jim",
      "last_name": "Larssen",
      "pronouns": "he/him/his",
      ...
    },
    "contacts": [
      { "relationship": "colleague", "profile": { "called_name": "Joe", ... } },
      { "relationship": "manager", "profile": { "called_name": "Johanna", ... } }
    ],
    "attendances": [
      { "name": "Stand-up", type: "meeting", "cadence": "daily", ... },
      { "name": "Company Kick-off", type: "company event", time: "10:00", ... }
    ]
  },
  "sender": {
    "called_name": "Joe",
    "first_name": "Joe",
    "first_name": "Schmoe",
    "pronouns": "he/him/his",
    ...
  },
  "contacts": [
    { relationship: "colleague", profile: { "called_name": "Jimmy", ... } },

```

```

    { relationship: "manager", profile: { "called_name": "Johanna", ... } }
  ],
  "text_snippets": [
    { "type": "tweet", "content": "..." },
    { "type": "twitter_comment_reply", "content": "..." },
  ],
  "attendances": [
    { "name": "Stand-up", type: "meeting", "cadence": "daily", ... },
    { "name": "Company Kick-off", type: "company event", time: "10:00", ... }
  ]
},
}
}

```

Note: This is set up with experimentation in mind so that we can quickly swap out prompt templates with new ones and manually experiment with campaigns.

Fuzzy preamble from notes (outdated)

Scraping

Given the fact that scraping is an incremental process towards better and better, the data and the scraping itself should be fairly separate, i.e. let's not couple scraping with storage (I know this is vague, but we should keep the intent in mind). This is because I expect scraping to be quite bad at first, so it should be easy to ingest data from other (non-scraping) sources, for example:

- Buying data from e.g. apollo.io
- Manual input (adding data manually through e.g. a Django admin panel)

Email formulation + sending

Sending an email involves, in the end, a bunch of data, the fact that we are using an LLM to formulate it is an implementation detail. To me, the question is: does the logic have access to the data/db or is it more like a payload based on which an email is formulated (and maybe a separate service, etc.).

Some decisions about the email need to take place (in what language is the email? Who's getting impersonated, and who is being targeted? At what time will the email be sent? If the email formulation + sending logic is based on a payload, meaning that that logic doesn't have "raw" access to the data, there will always be parts of the logic (arguably the most important ones) that need to live in the sender of said payload rather than the receiver.

With this said the logic for fishing formulation + sending (be it a separate microservice or just a module in a codebase), will:

- expose an endpoint/function for each phishing attempt type (email, sms, etc., currently only email)
- said endpoint/function will accept a payload/arguments which will determine the email shape and content

Open question: we'll want to imitate a company email css, images etc. Where is this information stored? Is it passed along the "payload"? Images would need to be hosted somewhere, which implies that either the caller of this logic hosts them or it will need to pass the images along (or other complex alternatives)

User-facing application/dashboard

The equivalent of the BFF for NaturalQuery.

"You have been pwned" pages

Let's say a user decides to follow a link we provided and inputs data in a form. On submit, it should be led to a "you have been pwned" page which explains what happened, etc. etc.

Something similar should happen for emails. The purpose for this would be to avoid storing other companies secrets when a user falls for the phishing attack.

Profile data

Where is the actual data stored? What I mean: to which service does it belong? That implies access patterns to said data.

If the dashboard owns said data, will the scraper POST the data to the dashboard for storage? Or scraping is done by the dashboard application? (The latter feels strange)

Loot data

Given the different types of phishing attacks, there are different channels through which the “loot” could go through: email, a view with a form, etc. This means that there needs to be a service in charge of checking if any data arrives from these channels (periodically or reactively, i.e. on form submission). Example:

- we ask person X for some password
- the password is put in a form we linked in our phishing email and submitted
- this event needs to be acknowledged and kept track of

Attack Service