# Attack Service

## Domain boundary

Receives: Objective

Fetches: Target, Organization

Execute: Attack

Writes: Attack, AttackLog

Copies: Objective

## Flow

1. Receive a POST request of Objective.
2. From the request payload of the request, extract the required info and pass it along to the ProfileService. Receptionist creates the initial attack per target accordingly.
    a. Store `campaign.objective` for reference.
    b. Retrieve `individual_id` of the targets: POST `org_id` , the `email` s of the targets, and `goal` to ProfileService to get the profile data of these targets.

    Payload

    ```
    {
      org_id: string,
      goal: string, // Objective.goal; e.g. click on the malicious link.
      targets: {email, ...social links} []
    }
    ```

    Response

    ```
    {
      targets: {
        individual_id: string,
        email: string
    ```

```
  }[]
}
```

c. Create the first Attack of this campaign per target, initial `status` is `WAITING_FOR_DATA`.

d. Return attacks back to Dashboard.

```
{
  attacks: { attack_id: string, individual_id:string, email:string } []
}
```

3. AttackCoordinator checks and updates `attack.status` periodically.

   - `WAITING_FOR_DATA` : poll ProfileService to check if ProfileService scraped the required information for the target. If true or ~~the objective is going to expire in [3 days]~~, save `attack.target` and set `attack.status` as `ONGOING`.
   - ~~`PENDING` : with the given `attack.target` , create a phishing email with a link and a tracking pixel using `attack.id` and save it as `attack.artifact` . At this stage, we can fine-tune the content or set `attack.status` as `WAITING_FOR_DATA` if more information is required. If the phishing email is considered ready, set `attack.status` as `READY` (via an API call or Django Admin)~~
   - ~~`READY` : send the email. At the moment we only have one actionable. For various actionable types, we might need an AttackAgent to perform the task.~~
   - `ONGOING`:
     - - if Attack has no artifact, create the first artifact.
     - - if Attack has approved artifacts, deliver them.
     - - if no deliverables, check if Attack is expired. Get the latest `AttackLog` by `attack_id`. If the timestamp of the latest AttackLog is older than [3 days], set `attack.status` as `FAILED`.
   - `FAILED`: Initiate a new attack if there's enough time left in the campaign.
   - `SUCCESS`: do nothing.

4. AttackEventListener listens to incoming events for attacks (only `ONGOING` attacks can have incoming events). If the event is considered a success, ask AttackCoordinator to set the status of the attack as `SUCCESS`.
5. AttackLogger logs the following events:
    - `EMAIL_SENT`
    - `TARGET_OPEN_EMAIL` : [nice to have] user opened an email with a tracking pixel.
    - `TARGET_CLICKED_LINK`
    - `TARGET_` `SUBMITTED_CREDENTIALS` : [nice to have] if we offer to create a malicious website.
    - `ATTACK_EXPIRED`
6. ObjectiveCoordinator checks if an objective has expired. If true, set all the `ONGOING` attacks as `FAILURE`.

## Components

**Receptionist:** take the incoming campaign and register the objective and create the initial attacks accordingly.

**ObjectiveCoordinator:** [async] monitor and update the status of an objective.

**AttackCoordinator:** [async] monitor and update the status of attacks to the end of the objective.

**AttackEventListener**: listen to external events (e.g.`TARGET_CLICKED_LINK`) and pass them to AttackCoordinator timely.

**AttackLogger:** create AttackLogs per outgoing/incoming event.

## Attack playbook

It's sketchy to expect ProfileService to come up with an arbitrary "data quality" score before we build up knowledge in this area. At the moment, we will take a more pragmatic approach as follows.

AttackCoordinator requires a playbook to determine what information to query and whether it has obtained all the necessary data to perform an attack. It might not be the best categorization, but let's brainstorm some ideas. The “degree” is inspired by the theory of the six degrees of separation.

The tricky part is that it is difficult to predict the required time. We can only perform the best-possible attack based on a rough estimate. We always try to perform higher-degree attacks but may fall back to a lower-degree attack due to time constraints. The time required per category is subject to our scraping skills.

For example, if there are only 8 days left and we cannot scrape any information about associates or affiliates, we lower our criteria to the 2nd degree. If there are only 3 days left, we resort to the 1st-degree attack. Happenings have a higher weight due to their nature. Once any happening is scraped, a timed attack can be launched, regardless of whether other complementary data is present or not.

## 0degree (remaining time ≥3 days)

Criteria: [name & email & org & location]

## 1 degree (remaining time ≥ 7 days)

Criteria:  [name & email & org & location], [department, peers]

## 2 degrees (remaining time ≥ 11 days)

Criteria:  [name & email & org & location], [department, peers], [associate, affiliate]

## Timed

Criteria:  [name & email & org & location], [happening]

```
classDiagram
direction LR

class Objective~AttackService~ {
  +String       id
  +String       goal
  +DateTime     begins_at
  +DateTime     expires_at
  +String       org_id
}

class Attack~AttackService~ {
  +String     id
  +Objective  objective
  +String     status
  +JSON       target
  +JSON       content
  ...
}

class AttackLog~AttackService~ {
  +String attack_log_id
  +String attack_id
  ...
}

class Organization~ProfileService~ {
  +String org_id
  ...
}

class Individual~ProfileService~ {
  +String id
  +String org_id
  +String email
  ...
}

Objective --> Attack : objective_id
Organization .. Attack : org_id
AttackLog --> Attack : id
Individual .. Objective : target
Individual : org_id .. Organization
Individual <-- Attack : individual_id
```

## End 2 end integration

These items are Must-Have’s. Let’s aim to finish these and then add other nice-to-have’s.

- [x] ~~[Dashboard] create and launch a campaign.~~
- [x] ~~[AttackService] create an Objective and an Attack per target~~
- [x] ~~[Dashboard] poll the ongoing Objective status from AttackService~~
- [x] ~~[AttackService] poll ProfileData of the targets from ProfileDataService~~
- [x] ~~[AttackService] actually use the llm to generate the email~~
- [x] ~~[AttackService] create an Email once profile data is available~~
- [x] ~~[AttackService] request approval for attack artifacts requiring it~~
- [x] ~~[AttackService] send the approved Emails.~~
- [x] ~~[AttackService] listens to the click events from the emails. Update the Attack status~~
- [x] ~~[Dashboard] shows the updated Attack status correctly, i.e. link clicked.~~
- [x] ~~[AttackService] end the objective correctly.~~
- [ ] [Dashboard] end the campaign correctly.

Nice to haves

- [ ] go through some of the TODOs in the codebase
- [x] ~~[AttackService] https://linear.app/orchest/issue/SEC-14/track-user-opened-email-event~~
- [x] ~~[AttackService] allow regeneration of attack artifacts that allow it~~

    https://github.com/orchest/molesec-attack-service/pull/19

- [ ] [ProfileDataService] Create a Profile per target if not existing.

```
https://linear.app/orchest/issue/SEC-26/profiledata
```

- [ ] [Dashboard] Campaign funnel diagram