Arabella

# Hackathon Notes

This project was conceived in the wake of the Social Network Hackathon. All the work on this project, investigation, documentation and coding is documented in the github repository at <link>. The history of what work was done when can be seen there in its entirety. As per hack rule and tradition, all coding was left for the period of the hack itself. However, much of the prep-work including documentation and design was pre-loaded (see the github history). If that negatively affects our hack score, c’est la guerre, we’re prepared to take the hit.

# System

In the course of the Bellingcat Social Network Hackathon several data points came together to inspire a system proposal.

1. In the Bellingcat survey of OSINT investigators (link), X% of respondents reported being frustrated in the use of FOSS tools on github. Reasons reported included things like “tool doesn’t work at all”, “no documentation” and “doesn’t produce the output I need”.
2. A quick, random survey of a half-dozen open source tools\* produced the anecdata that two\* of the tools didn’t work at all, but could be (and were) fixed by a professional coder with a couple of lines of code in less than 2 hours of work.
3. In networking with the other participants, it became apparent that there is an untapped set of professional and semi professional coders and data scientists that:
   1. want to contribute to the work of OSINT but don’t know what needs to be done that matches their skill set

OR

* 1. want to career-switch to OSINT but don’t have a way in to the field

## Goals:

1. Solve tool problems for OSINT investigators in near real-time
2. Build a ranked set of FOSS tools that can be watched and proactively kept up to date
3. Give Bellingcat a way to tap the expertise of a wide range of people who want to contribute but have no way currently to do so
4. Build a “bench” of people that Bellingcat can draw on when recruiting for internal positions
5. Give back to FOSS community by PR’ing any tool fixes back to the original builder
6. Build a set of tools that are known-to-work and known-to-be-useful by Bellingcat investigators

## Risks:

1. Supply chain attacks
2. System maintenance cost/effort
3. System operational cost/effort
4. Participant vetting cost/effort

## Mitigations:

1. Investigator and Fixer remain unknown and unknowable to each other
2. Vetting of participants
3. Expectation that this is a store with operating hours, not a 24/7/365 unattended automat.

## Investigator Vetting (IVet)

We expect that investigator vetting is an entirely offline process handled separately by Bellingcat. All that Arabella needs to know is that an investigator has, or has not, been vetted. IVet is a single screen, reachable only through hardware key. Addition of IVet hardware keys is a manual process.

## Fixer Reputation (FRep)

As the system is designed to pull contributors not currently known to us into the system, FRep must be part of the system and allow contributors with a reputation of zero to make meaningful contributions. FRep is a combination of feedback provided in Fix Vetting (FVet) and Fix Acceptance (Facc). Vetters either accept or reject a fix, and if they accept they provide a numeric score for the quality and difficulty of the fix. That score feeds into Fixer Reputation. When an investigator accepts a fix, they provide a single quality score. Those feedbacks are raw data while how that data is reflected in FRep is an ongoing algorithmic exercise.

## Fix Vetting

Fixes are vetted through a 2-step process using traditional Git pull request process. A project being worked on is forked to a PRIVATE repository and 3 branches are created - main, develop and feature. The Fixer may only commit to feature.

A PR down to the develop branch on the forked repo is accepted or rejected with comment by the Bellingcat team. An accepted PR to develop triggers an automated PR down to main.

An accepted PR down to main automatically sets the repo visibility to public and triggers a notification to the original requester.

## PR Back

The PR Back function automatically creates a pull-request back to the originating repo to give the original tool author and his users a chance to benefit from the fix. Feedback from the PR Back process is added to the FRep score - with “accepted without comment” being the highest score.