Threat Detector®

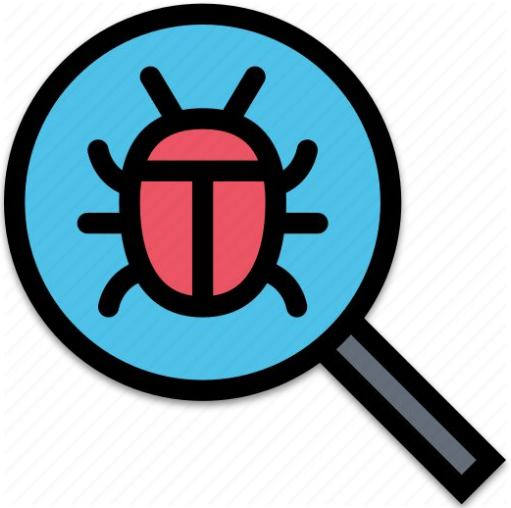
v1.1.0

Team H

Mark Biegel, Tim Yingling, Jenna Pasternak, Scott Boyd, Clement Onoja, Biruk Yimer

What is it?

- Virus hash and URL scanner
- Versatile and robust application
- Uses VirusTotal API for scanning capabilities



Why Threat Detector is needed

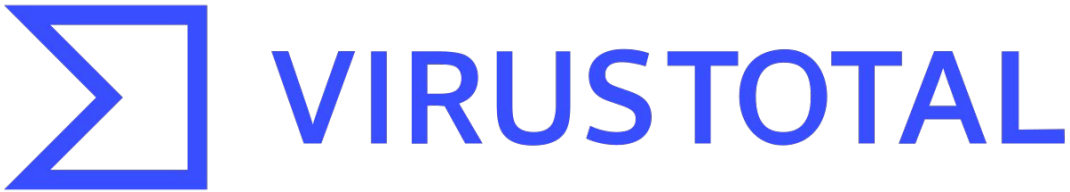
- Need to be able to scan file hashes to see if they are malicious
    - Enables users to see if a file is malicious before using or executing it

- Scan websites to see if they contain malicious content
    - Enables users to see if a site is malicious before visiting it

Motivation

- Create a website similar to VirusTotal
  - Main goal: report and store status of given file hashes
  - Extra feature: report and store status of URLs

- Allow for cohesive user experience between scanning hashes and URLs

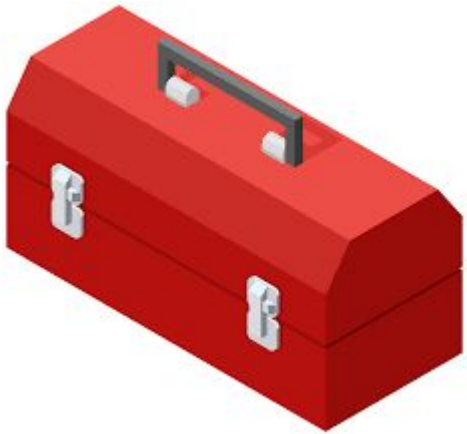


Our Tools of Choice

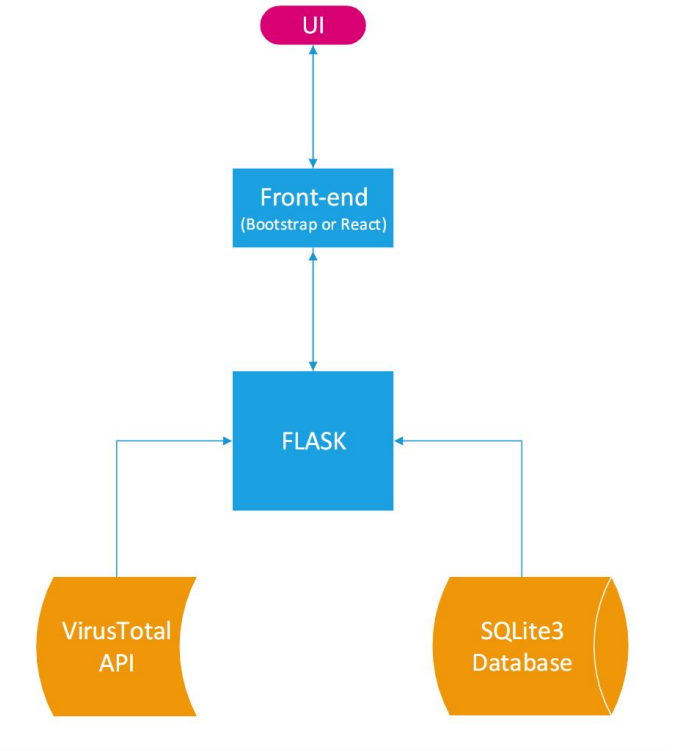
- Chose to use tools we made ourselves familiar with during the homework
  - SQLite3
  - Flask
  - Python

- Had to make our front end look nicer than plain HTML
  - Bootstrap

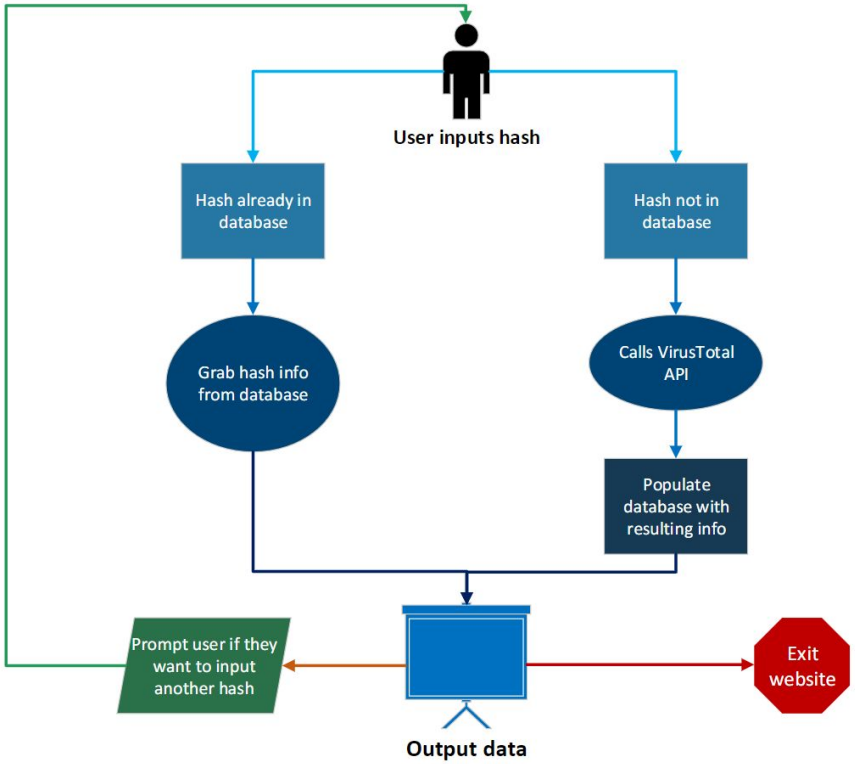
- Organization
  - Github
  - Jira



Starting our Design Process in Sprint 1: Architectural Design



Average Use Case: Activity Diagram



Starting Development: Splitting up the Work for Sprint 2

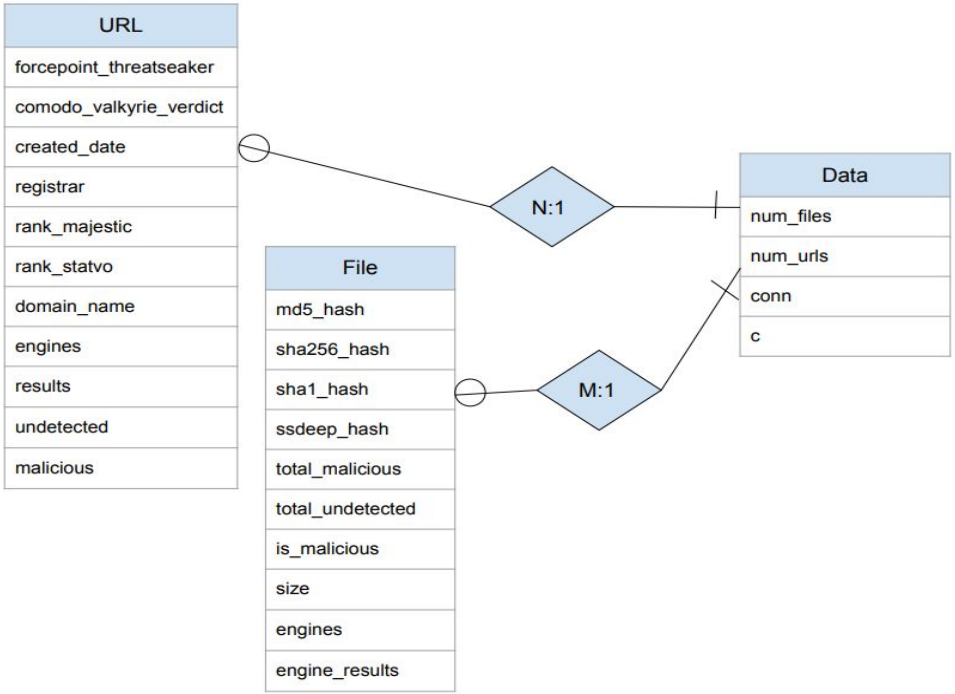
- Split up responsibilities into three categories
  - Database: Tim Yingling and Jenna Pasternak
  - Back end: Clement Onoja and Scott Boyd
  - Front end: Mark Biegel and Biruk Yimer

- Roles were not strict

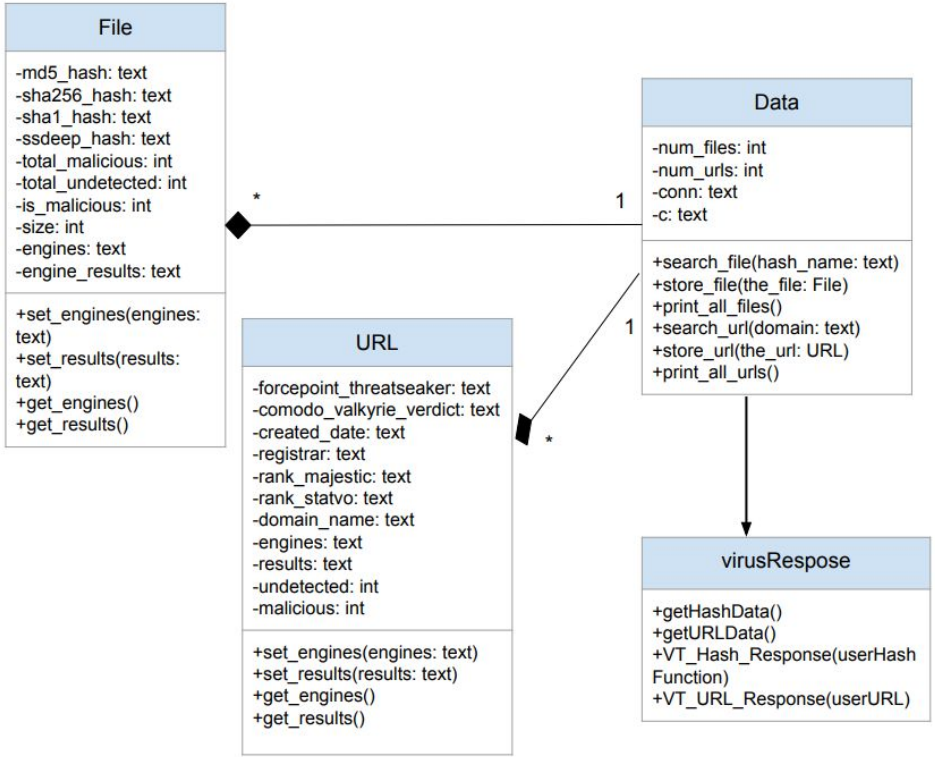
Database

- Created 3 classes for database functionality
    - File
        - Container for file hashes
    - URL
        - Container for URLs
    - Data
        - Store and Search

ER Diagram


URL
forcepoint_threatseaker
comodo_valkyrie_verdict
created_date
registrar
rank_majestic
rank_statvo
domain_name
engines
results
undetected
malicious
N:1
Data
num_files
num_urls
conn
c
File
md5_hash
sha256_hash
sha1_hash
ssdeep_hash
total_malicious
total_undetected
is_malicious
size
engines
engine_results
M:1

Class Diagram



Back End: Hashes

- Created VirusResponse class to handle working with the API
- User can input the hash in the following formats:
  - MD5
  - SHA256
  - SHA1
  - SSDEEP
- VirusTotal retrieves other 3 hash types as well as additional information to store in the database

Back End: URLs

- Uses a separate API call to retrieve information about URLs
- Most of the information is different from file hashes, but engines and results are still retrieved
- Challenges:
  - Differentiate between an entered file hash and URL
  - Limitations of VirusTotal Free API Key
    - Required the user's input to be "cleaned"
  - Different keywords for malicious vs clean URL

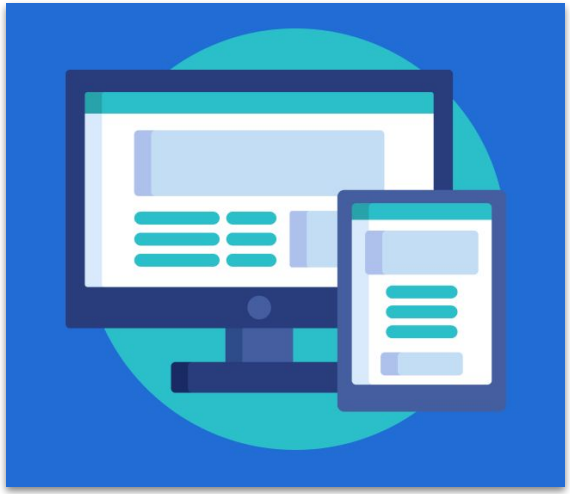
Front-End

User Experience is the forefront of the likability of a product

A cohesive interface is the foundation for a quality user experience.

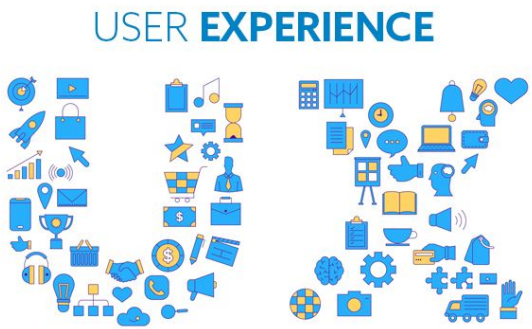
Threat Detector's front-end goals:

- Cohesive layout from page to page
- Scalable on multiple devices
- Easy-to-navigate
- Simplistic, yet informative
- Intuitive actions and responses
- Modern and sleek

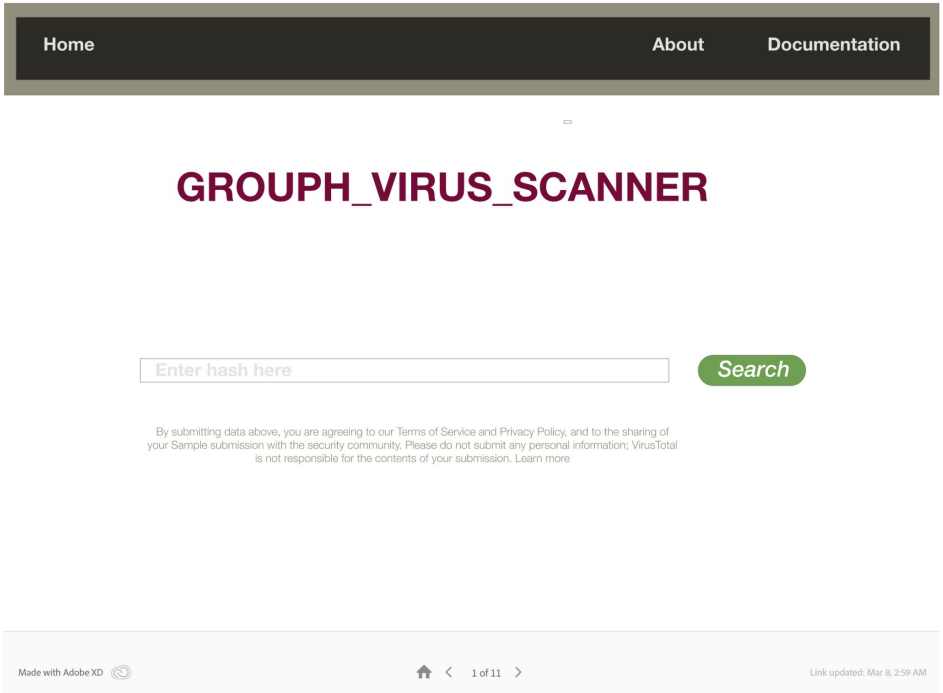


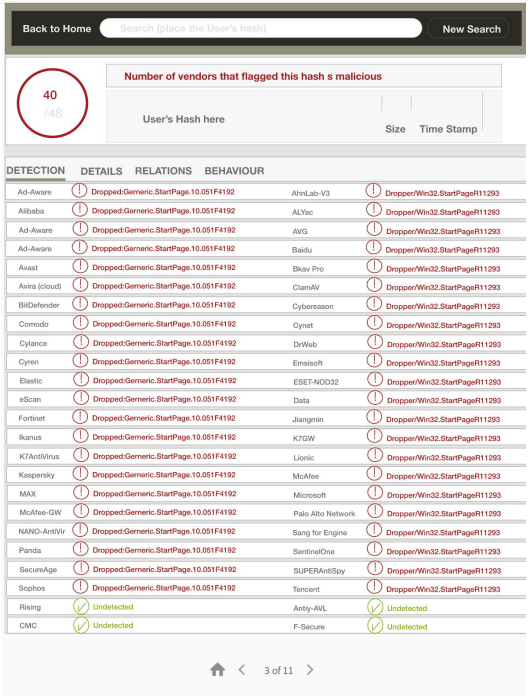
Front-End - User Experience Tactics

- Rich, Pastel Colors
- UI Object shapes and sizes
- Graphics-
  - High resolution images
  - GIFs
  - SVGs
- Thin, crisp fonts
- Output information sized larger or smaller based on importance of information



Front-End - Initial Mock-up





Mock up of potential front-end's look and feel made in Adobe XD

Front-End - Layout

Bootstrap Framework with CSS for easy implementation and cohesive appearance

Front-end composed of 8 pages:

- Homepage
- About
- Help
- Invalid Entry
- Malicious Hash and URL (x2)
- Clean Hash and URL (x2)

Testing Our Product

- Tested code as it was developed, but wrote formal tests once the product was complete
- White box testing
    - Most of the database functionality
- Black box testing
    - Testing the product as a whole

Database White Box Testing

- Created a single test file: data_test.py
  - Started tests on an empty database file
  - Made sure that search and store functions worked correctly
- All database tests passed without having to modify the code

Front-end Black Box Testing

- Testing was performed by using different inputs
- Main focuses:
  - Displaying the correct status for hashes and URLs
  - Invalid inputs
  - Routing works correctly

Important Documentation

All documents are available in the Github repository

- README for our source code
- Administrator Manual
- System Testing Documentation

Demo Time!

Lessons Learned

- Learned how to use GitHub
  - Maybe not the best use of it, but learned how to separate development and merge code
- Also learned how to use organization software with Jira
- Being thrown into web development was a good way to learn
  - Not fun in the moment, but overall beneficial
- Learned that we shouldn't be calling the API 10 times per run
  - Our first major "bug"

If We Were to do it Again...

- Plan out the extra feature earlier
- Have more meetings to explain developmental decisions
    - Communication was a weak point
- Have a better foundation for the project in our sprint 1
    - Didn't translate as well in our sprint 2
- Consolidate front-end HTML pages

Computer Science 447 Overview

Benefits:

- Homework 1 was VERY helpful (being thrown into it was helpful)
- Gained experience working with a team

Areas for Improvement:

- Better communication from professor to teams for what is expected of them
    - More discrete instructions
- Alternative software to Jira (i.e Github Projects)

Questions?