HackTrack

Project Plan

Team Dec1703

Dr. Daniels/Dr. Daniels

Team Members & Roles

Davis Batten - Key Concept Holder

Vitalie Cernetchi - Communications

Daniel Doyle - Team Leader

Nicholas Lewis - Communications

Anh Nguyen - Webmaster

Team Email

dec1703@iastate.edu

Team Website

http://dec1703.sd.ece.iastate.edu

Revised 4/26/17 - v2.1

1 Introduction

1.1 Project statement

An attack graph is graphical representation of a secured computer system and all the paths through it that an attacker may take to achieve their goals. Often these graphs are produced by hand[1]. This project is an effort to create an attack graph-based web application to assist the red team of Iowa State's Cyber Defense Competitions. An admin user will sketch out the base structure of the blue team's system, then the red team members are free to enter and record various ways they have attempted or been successful in attacking specific team's systems. The attack graph displayed by this information will be dynamic and react to changes and create predictions for other red team members to use to their benefit.

1.2 Purpose

The project will enable many members of a red team who do not know one another, or aren't working together, to communicate successful attack strategies to others. It uses the progress of others to further your own and prevent the same discoveries from being repeated. Such a system would be valuable to the red team over their current model, a wiki, because it won't require extensive effort or data entry to communicate success.

1.3 Goals

- Have a working product by the Fall Cyber Defense Competition
- Stable database server running Neo4j
- The system will automatically suggest possible tasks for red team members
- Design a variety of views to better show data
    - This goal is beyond the main product requirements, and so we consider it optional

2 Deliverables

- Source code for the application
- Full documentation on how the application behaves
- Virtual machine image implementing the project

[1] http://pages.cs.wisc.edu/~jha/jha-papers/security/CSFW_2002_1.pdf

3 Design

As the main deliverable of the project is a web application, our design relies upon several standard paradigms in web technologies. Almost any modern web application is built upon a client-server architecture. However, there are several technology options available for both the client and the server.

There are no shortage of available web front-end frameworks. We looked at several such frameworks, including Angular, React, and Ember. The main goal of this component would be to support dynamic UI updates as well as attractive visuals.

For the server, we evaluated several languages/frameworks. These included PHP, Ruby, .NET, Python, and Node.js. Beyond the basic server framework, another key component of the back-end of our application is the database. Dr. Daniels' research with attack graphs led to the natural conclusion that our app should utilize a graph database. There are several graph databases on the market, including Neo4j, OrientDB, and ArangoDB. The graph database chosen has a large impact on our decision of server framework.

Beyond the simple client-server paradigm, we knew early on that our application would need to be capable of real-time communications. This requirement meant we would need some form of real-time messaging between client and server. Each of the server frameworks evaluated came with its own set of real-time frameworks, most involving the use of websockets. This lead to our basic software architecture as shown in Figure 1.

3.1 Previous work/literature

Dr. Daniels gave us a paper called *An Annotated Review of Past Papers on Attack Graphs*[2]. This is a comprehensive listing, with findings, of previous research that has been done on and done about attack graphs up to March 2005.

A section of this paper goes into detail on how to "analyze alerts from intrusion detection systems" (p. iii) which is a portion of our final project. The reachability problem focuses more on the time complexity of this issue, but the system can also be used to generate recommendations for the next steps to take.

3.2 Proposed System Block Diagram

The following block diagram (Figure 1) shows our proposed system architecture. It will be modeled as a standard client-server application. The server will handle all requests to the database and serve data and views to the waiting clients. This server will also be connected to a graph database to store application data and the attack graphs.

[2] http://www.dtic.mil/dtic/tr/fulltext/u2/a431826.pdf

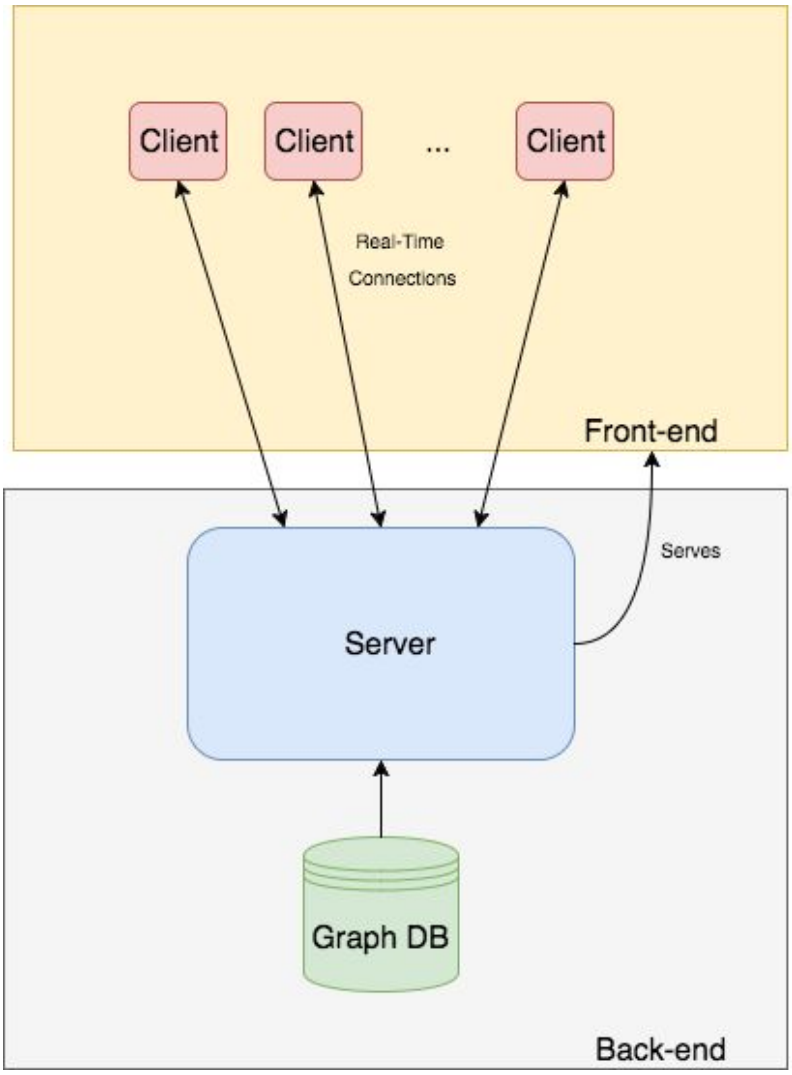

Figure 1: Basic Software Architecture

3.3 Assessment of Proposed methods

When evaluating candidates for our front-end framework we found that most are based upon the idea of view components. However, how those frameworks went about that component model very differently. React uses JSX (Javascript that compiles to HTML) and relies upon several third party libraries and packages. Angular is complete framework that utilizes Typescript. Additionally, several members of our team have used AngularJS, the precursor to modern Angular. Ember does not follow the component concept as closely and relies upon data-binding tools like Handlebars instead.

Our choice of graph databases was made for us by our advisor/client Dr. Daniels. He specifically asked us to use Neo4j as that is currently the most popular (and most supported)

graph database available. The other graph databases were not evaluated by our team.

The decision to use Neo4j as a graph database complicated our choice of server framework since it has support for so many different languages. Choosing the server language ultimately became one of personal preference. The team decided on a framework that some members had prior experience with and minimized the learning curve for those who did not have experience.

After evaluating our options for each of the following major software architecture components, we settled upon a solution using Angular, Node.js, and Neo4j. Because we chose to use Node.js as our back-end framework, we are also using two of the most common packages for Node.js: Express and Socket.IO. These packages allow us to properly serve front-end files and perform real-time updates. A block diagram showing the overall architecture for our application can be found in Figure 2.

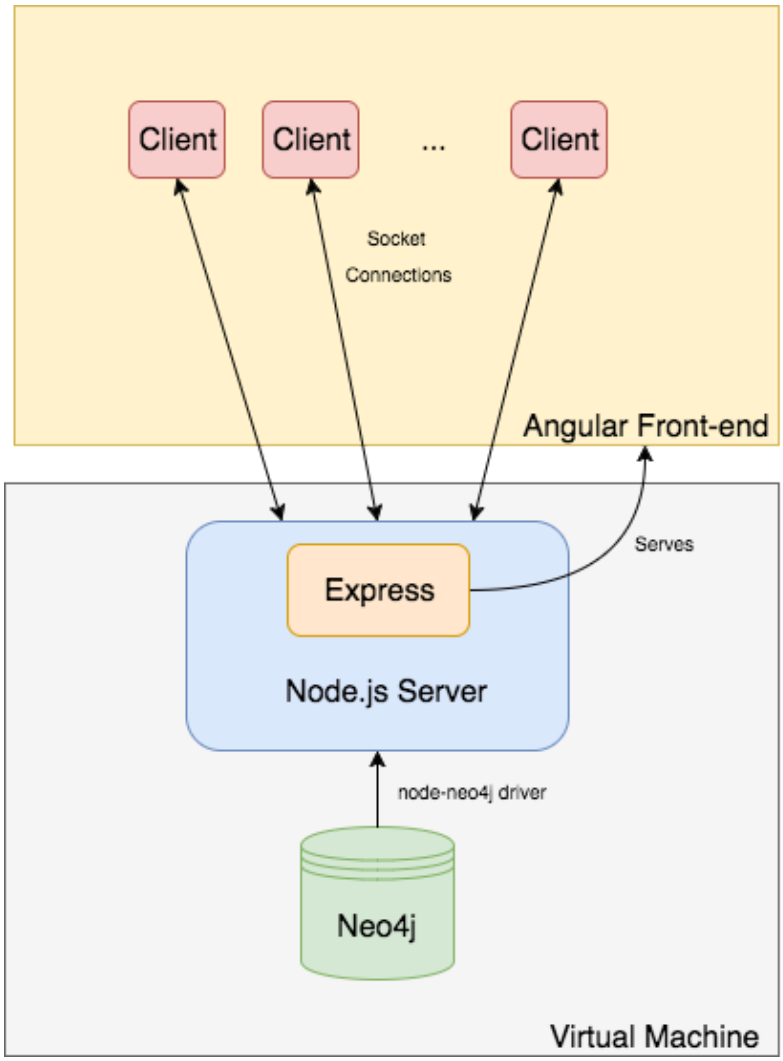

Figure 2: Detailed Software Architecture

3.4 Validation

Validation of our solutions will be accomplished in several ways.

1. While writing code, we will manually test the functionality we are working on.
2. We will use unit tests and end-to-end tests.
3. We hope to have members of IASG, our target audience, do some rudimentary usability testing during the second semester.
4. The final test will be a trial run during the 2017 Fall Cyber Defense Competition. This will be a strenuous test at scale.

In order to confirm the completion of the project requirements as defined in the Section 4 (Project Requirements/Specifications), the following testing will be specifically performed:

- A user will create a new account
- The user will login to this new account
- The user will access data about a given Blue Team
- The user will follow/subscribe to that Blue Team
- The user will perform an edit to that Blue Team's graph
- The new edit should trigger a hypothesis alert and add it to the user's task list
- The user will "test" this hypothesis, and confirm it
- Upon hypothesis confirmation, the server should find a path to a flag and alert the user
- The user will claim the flag
- A second user will try to concurrently change data on the same graph as the first user
- The server logs for the test will be output

4 Project Requirements/Specifications

4.1 Functional

The following list of functional requirements were discussed and confirmed with our client, Dr. Daniels.

- Users can sign in
    - Users can be authenticated into the application using a password
- Accounts have a name and a picture
- Standard user and Red Team Leader account types
    - Standard users represent most participants on the Red Team
    - Red Team Leaders act as admins and have several exclusive features
- Users can view all data about Blue Teams
    - This data should be able to be filtered by team
- Simple account creation
    - Two implementations of this requirement exist (choose one):
        - Utilize the existing ISEAGE user account registration system
        - Self-registration pending approval from a Red Team Leader
- Red Team Leaders can create/edit the master scenario
    - They should be able to use the scenario creation method being created by one of Dr. Daniels' graduate students.
- Users can edit a Blue Team attack graph
    - This will include creating and editing nodes and edges in the graph
- Hypothesis edges created from found edges
    - When an edge is found for one Blue Team, hypothesis edges should be added for all other Blue Teams
- Server checks for path to flag
    - The server should be capable of finding complete paths to flag nodes in the graph and should alert the users to this path

- Ability to claim/plant a flag
    - Users should be able to fill out a form upon the successful finding/planting of a flag during the competition
- Users can follow a Blue Team
    - This will subscribe the users to relevant updates about that team
- Should track updates by user
    - The server should log all updates to graphs by users, and this information should be available to Red Team Leaders (and possibly standard users)
- Task list should be maintained for each user
    - A list of tasks to be completed by each user will exist and each task will be given a specific priority depending upon several factors

4.2 Non-functional

- Multi-browser
    - Support for the most popular browsers (Firefox, Chrome, and possibly Safari)
- Secured against Blue Team members
- Perform standard logging operations on the server to aid the admins and developers
    - Should include changes to the database, edits to the scenario, etc.
- Hosted by virtual machines
    - Current implementation calls for only one VM, but using two VMs is also acceptable
- Minimum of 50 users supported concurrently

4.3 Standards

The web application will strive to meet several standards for web applications. Developers will be held to coding best practices such as comments, naming conventions, and flexible code. The application will also follow the web design standards as prescribed by W3C for HTML, Javascript, and CSS[3]. Due to time constraints, we may forego the accessibility web standards from W3C. This is potentially unethical as it means that users with disabilities may not be able to use the application fully.

The software architecture that we settled upon in the design phase also meets several

[3] https://www.w3.org/standards/webdesign/

standards. Most of the architectural components are very popular with web developers and de-facto standards amongst the community. This means that support will be easier to obtain and access to a large ecosystem of plugins and packages. These technologies include Angular, Node.js, and Neo4j. Much of our architecture is built upon the ideas behind the MEAN stack, which uses MongoDB, Express, AngularJS, and Node.js to rapidly prototype web applications[4]. Our solution substitutes Neo4j for the NoSQL database, and the more current Angular for AngularJS. Interestingly, this architecture is something of a proposed standard, making up what is known as the ANNE stack (Angular, Node.js, Neo4j, Express)[5].

5 Challenges

The database schema from Dr. Daniels has yet to be decided. This schema is a prerequisite to begin any major work on hypothesis edges or flag path detection. Until we solidify the schema, we can only work on peripheral functional requirements like user authentication.

In the current system, data is copied and pasted as new information is discovered. This leads to a very unorganized data dump, which makes it very difficult to read and causes some discoveries to be repeated. Our goal will be to provide the users with tools that will catalog their discoveries in an organized fashion and allow for quick communication between team members whenever a discovery is made. However, one challenge might be that the Red Team members may be stubborn and prefer to not give our new system a chance, instead relying on the current system.

Another major challenge will be usability, to make sure that the users are able to navigate with ease and doesn't hinder their work while interacting with the application. There will not be time to teach the users how to use the software, so it must be intuitive enough to learn on the fly.

An issue that we foresee arising involves how database entries will be handled when multiple users are trying to access the same area. This issue has a few different options on how it should be handled, however, we will most likely wait until we are farther in the development process to make a decision.

[4] http://mean.io/
[5] http://www.42id.com/articles/anne-stack-angular-js-node-neo4j-and-express/

6 Timeline

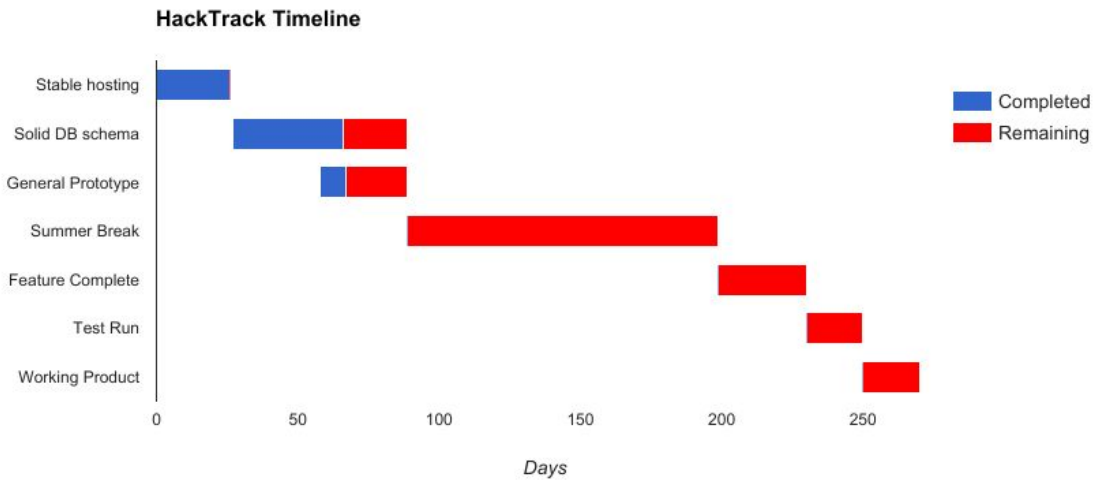

Figure 3: Gantt Chart

6.1 First Semester

- Stable Hosting of Virtual Machine and Database - Feb 28
    - Have the VM running with a functioning database on it
- Solid Database Schema - May 2
    - Define how the database will be maintained
- General Prototype (core features only) - May 2
    - Functioning web interface with database interaction

6.2 Second Semester

- Feature Complete - Sep 20
    - Remaining requirements fulfilled
- Test Run with Sample Users - Oct 10
    - Run through of system features to ensure correctness
- Working Product - Oct 30
    - Project is deemed complete and ready for deployment

7 Conclusions

The goal of the project is to build a multi-user, web-based management tool for Red Teams in Cyber Defense Competitions. The system will support red team users through graph data structures to evaluate blue teams. Red team members will be able to record attempted attacks and create hypothesis alerts to speed up the discoveries process.

- Feb 2 - Feb 28: Stable hosting of Virtual Machine and Database
- March 1 - May 2: Solid Database schema.
- April 1 - May 2: General Site with front-to-back functionality
- Aug 20 - Sep 1: Prototype with all major components
- Sep 2 - Sep 14: Test run with Sample Users/Feature complete
- Sep 14 - Oct 1: Working product for Fall 2017 CDC

8 References

- Neo4j Guidelines for Developer
  - https://neo4j.com/developer/get-started/
- Graph Database
  - https://neo4j.com/developer/graph-database/
- Web socket.io tutorial
  - https://www.tutorialspoint.com/socket.io/index.htm
- Node.js
  - https://www.tutorialspoint.com/nodejs/
- Express Documentation
  - http://expressjs.com/
- Angular Documentation
  - https://angular.io/docs/ts/latest/
- An Annotated Review of Past Papers on Attack Graphs (Lippmann & Ingols)
  - http://www.dtic.mil/dtic/tr/fulltext/u2/a431826.pdf
- Two Formal Analyses of Attack Graphs (Sheyner, Wing & Jha)
  - http://pages.cs.wisc.edu/~jha/jha-papers/security/CSFW_2002_1.pdf

9 Appendices

Version	Date	Comment
v0.1	2/20/17	First draft
v1.0	2/24/17	First draft complete
v2.0	3/31/17	Second draft complete
v2.1	4/26/17	Updated timeline

Table 1: Version Chart