HackTrack

Design Document

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 - v1.1

Contents

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.

3.1 System specifications

Our system runs off of various frameworks and utilizes a select few software packages. The main website is running on version 7.6.0 of Node.js and serves with NPM version 4.1.2. NPM is used to manage add-ons like version 2.4.0 of Angular, version 4.14.1 of Express, and version 2.0.0 of the node-neo4j driver. The Node.js server interacts with a Neo4j graph database running version 3.1.1.

This is all hosted on a Virtual Machine that is running Ubuntu Server 16.04.2 LTS with four gigabytes of RAM and fifty gigabytes of storage.

3.1.1 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

3.1.2 Functional

- 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

3.1.3 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[2]. 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 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[3]. 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)[4].

3.2 Proposed Design/Method

We have decided to implement the aforementioned ANNE stack. There will be a Node.js server making queries to a Neo4j database which will use Express to serve content to an Angular based front-end website.

[2] https://www.w3.org/standards/webdesign/
[3] http://mean.io/
[4] http://www.42id.com/articles/anne-stack-angular-js-node-neo4j-and-express/

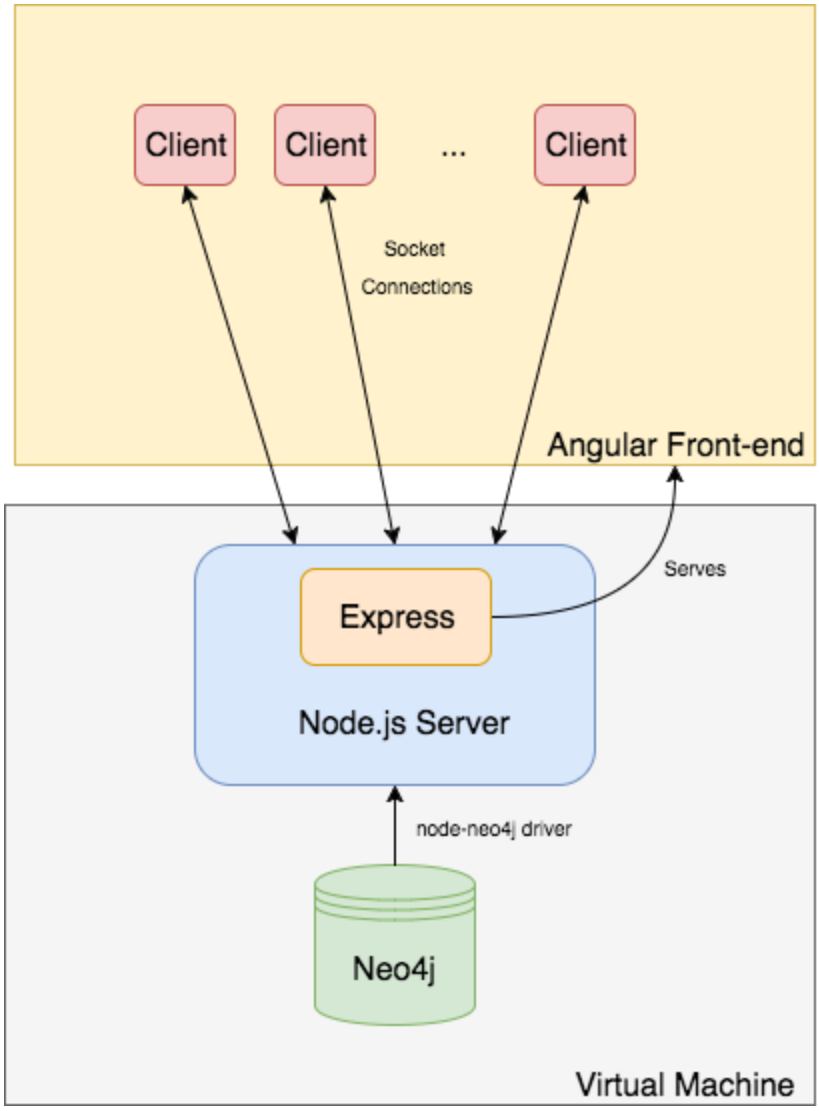

Figure 1: Detailed Software Architecture

The graphing database was chosen for us by Dr. Daniels. He specifically asked us to use Neo4j as it is well supported and commonly used. The server itself needs to work with Neo4j well, so we have decided to use the Node.js framework for the back-end to interface with the database and front-end. We are specifically using the node-neo4j driver to ease communications between the Node.js server and the Neo4j database. This driver makes it much simpler to make queries on the database from the JavaScript on the Node.js server.

3.3 Design Analysis

Our group has built a proof of concept for our architecture layout. The Virtual Machine is able to start up a web server and graph database which can interact successfully. This barebones prototype can be reached from an external machine and is able to load static content and make queries on an arbitrary database.

4 Testing/Development

4.1 Interface Specifications

Omitted for this version.

4.2 Hardware/Software

End to End Testing - Scripted actions that navigate to various pages and test for data accuracy. This is especially beneficial in the project as it shows that our project can pass data from the front end, through the server, into our database and vice versa.

Unit Testing - Testing individual route action to ensure data accuracy. This is important when considering the modularity of the system.

4.3 Process

We currently have a unit test for each route that performs a database action, and an end to end test script that ensures that the entire site can still be accessed. together these test methods ensure modularity and lessens the risk of individual changes affecting separate components. Our unit tests check for performance, security of data, and usability.

Front end method testing will include ensuring data privacy for security. When testing for usability and performance, we want to see quick reaction times to user input, and an obvious outcome to come from it.

For server and database method testing, the security tests will encompass encrypting password information as it passes through the system, as well as blocking injection from input. The queries to the database should be concise so our performance tests will cover that.

5 Results

At this time, our testing has been limited to a basic end to end test of the software architecture, as well as some unit tests for the routes we have designed for user actions. The successful result of these tests proves that our architecture design is feasible and that our data skeleton properly connects all of the architecture components for data flow. We have received a skeleton database schema from Dr. Daniels and will be implementing some basic traversals over the Summer.

6 Conclusions

So far we have developed a design for the software architecture of our product. We have begun to implement this architecture with code, and have produced a barebones template using this architecture for our project. This template code successfully passed an end to end test proving its viability and correctness. Because our solution relies on web standards, and is an adaptation of a common architecture stack, we believe it is the best choice for this project. Our solution will help us to successfully complete a working product with a stable database server by the Fall Cyber Defense Competition.

7 References

List any references used in the document. These are an essential part of your review so far.

- 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/
- MEAN Stack
  - http://mean.io/
- ANNE Stack
  - http://www.42id.com/articles/anne-stack-angular-js-node-neo4j-and-express/
- 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

- W3C Web Design and Application Standards
    - https://www.w3.org/standards/webdesign/

8 Appendices

Version	Date	Comment
v0.1	3/5/17	First draft
v1.0	3/10/17	First draft complete
v1.1	4/26/17	Updated testing and results

Table 1: Version Chart