HackTrack

FINAL REPORT

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 12/3/17 - v1.2

TABLE OF CONTENTS

1 Introduction

This chapter will define the project through a project statement (Section 1.1), describe the purpose of the project (Section 1.2), and list some goals we had for the project (Section 1.3). Finally it will list the deliverables as agreed upon with our client (Section 1.4).

1.1 Project Statement

Iowa State holds multiple Cyber Defense Competitions (CDC) every semester. At these events, groups of participants form into ***blue teams*** and set up a network according to some provided specifications. On this network, they hide flags in specified locations, and increase security. All of this preparation is to try and prevent the ***red team*** of hackers from gaining access and capturing the flags. 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 ***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. In our implementation, an admin user sketches out the base structure of the blue team's system in what we are calling the ***scenario graph***. From this, individual ***team graphs,*** representing each blue team, are created. Red team members are then free to enter and record various ways they have been successful in attacking specific team's systems on. The attack graph displayed by this information is dynamic and reacts to changes and creates predictions for other red team members to use to their benefit. The attack graph is stored in a ***graph database*** which stores information as nodes and edges in a graph as opposed to rows in a table like a traditional database.

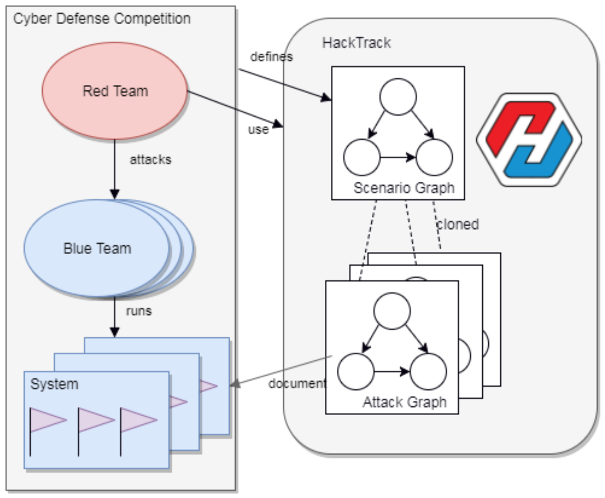

Figure 1: Usage of HackTrack within a CDC

1.2 Purpose

The project enables 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 the whole to further the individual and prevents members from spending time trying to do already completed work. Such a system is valuable to the red team over their current model, a wiki, as it doesn't require extensive communication or data entry to announce breakthroughs. When considering attack graphs, it is uncommon to find it used as a collaboration tool. In that respect, our project is considered a research project as well.

1.3 Goals

- Stable database server running Neo4j
- The system will allow red team members to enter new transitions they have found for a specific team
- A new transition will trigger hypotheses to be generated on other team graphs
- Hypotheses can be tested and confirmed by 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

1.4 Deliverables

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

2 Project Design

This chapter serves to discuss the design of the project. It will list the system specifications (Section 2.1) and specifically explore the requirements and standards necessary for project success. Section 2.1.1 lists requirements that did not change during the project, Section 2.1.2 lists requirements that were modified or altered during the development or design process, and Section 2.1.3 lists requirements that were removed or dropped.

Besides the system specifications, this chapter also describes in Section 2.2 the design process and some design decisions made. Finally Section 2.3 contains an analysis of the effectiveness of the final software design.

2.1 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 4.3.6 of Angular, version 4.14.1 of Express, and version 1.5.0 of the Neo4j driver. The Node.js server interacts with a Neo4j graph database running version 3.1.7.

We also use a Neo4j plugin to generate unique identifiers for our database objects. This plugin is version 3.1.4 of the GraphAware UUID plugin which also requires the same version of the GraphAware Core plugin. 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.

2.1.1 Requirements

Functional

- Users can sign in
    - Users can be authenticated into the application using a password.
- Users can view all data about blue teams
    - This data should be able to be filtered by team.
- Users can edit a blue team attack graph
    - This will include creating and editing nodes and transition edges in the graph.
- Hypothesis edges created from found edges
    - When a transition edge is found for one blue team, hypothesis edges should be added for all other blue teams that contain the same nodes as the endpoints of the edge.
- 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 record 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 as well as give a quick link to that team's page.
    - These updates include tasks assigned based on hypotheses or flag captures.

Non-functional

- 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.

2.1.2 Altered Requirements

Functional

- Simple account creation
  - Self-registration *does not* require approval from a red team leader.
  - This was changed to lessen the workload of the red team leader as well as simplify the process of registration since this is only a proof of concept.
- Red team leaders can create the master scenario
  - Red team leaders should be able to *use a Python Script* to initialize the scenario graph as well as setup an arbitrary number of blue teams.
  - We changed this because Dr. Daniels' graduate student did not create any tools for initialization of the master scenario.
- The server should log all updates to graphs by users, and this information should be available to admins
  - As it currently is implemented, it logs all updates by any user, but *does not keep logs on a per user basis*. We found this to be sufficiently detailed for our purposes.
- 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.*
  - All tasks hold the same priority.

Non-functional

- Support for the most popular browsers (Firefox, Chrome, ~~and possibly Safari~~)
  - We only support the two most popular red team browsers, Chrome and Firefox.
  - This was done to simplify testing.
- Minimum of 50 users supported concurrently
  - Technically the system can support a much higher number of users, however performance suffers.
  - We were unable to test this at scale, and have no concurrency protection.

2.1.3 Removed Requirements

Functional

- Standard user and red team leader account types
  - Account permissions and types were removed as they were deemed to have a low priority for our proof of concept.
- Accounts have a name and a picture
  - This metadata was removed from the project scope because it was deemed extraneous.

Non-functional

- Secured against blue team members
  - We were unable to complete this requirement under our time constraints.

2.1.4 Standards

The web application needed to meet several standards for web applications. Developers were held to coding best practices such as comments, naming conventions, and flexible code. The application also follows the web design standards as prescribed by W3C for HTML, Javascript, and CSS. Due to time constraints, we forewent 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. Our solution substitutes Neo4j for the NoSQL MongoDB 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).

2.2 Design Method

The first design decision we made was to opt for a graph database instead of a traditional SQL database. As the underlying concepts of our project are naturally represented in the form of nodes and edges, a graph database makes operations simpler and more efficient. The specific graph database framework was chosen for us by our client and advisor Dr. Daniels. He specifically asked us to use Neo4j as it is well supported and commonly used.

We then decided to implement the aforementioned ANNE stack (see Section 2.1.4). There is a single Node.js server making queries to a Neo4j database which will use Express to serve content to an Angular-based front-end web application.

The server itself needs to work with Neo4j well, so we decided to use the Node.js framework for the back-end to interface with the database and front-end. We are specifically using officially supported the neo4j-js 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.

The last design decision was to include a Neo4j plugin by GraphAware to generate UUIDs (universally unique identifiers). This was because we ran into issues with the dynamic nature of internal IDs in Neo4j. In order to have a static ID that could be referenced we customized the UUID generator from the plugin and added to our Neo4j database. This automatically adds UUIDs to all nodes and some specific key edges in the database upon creation.

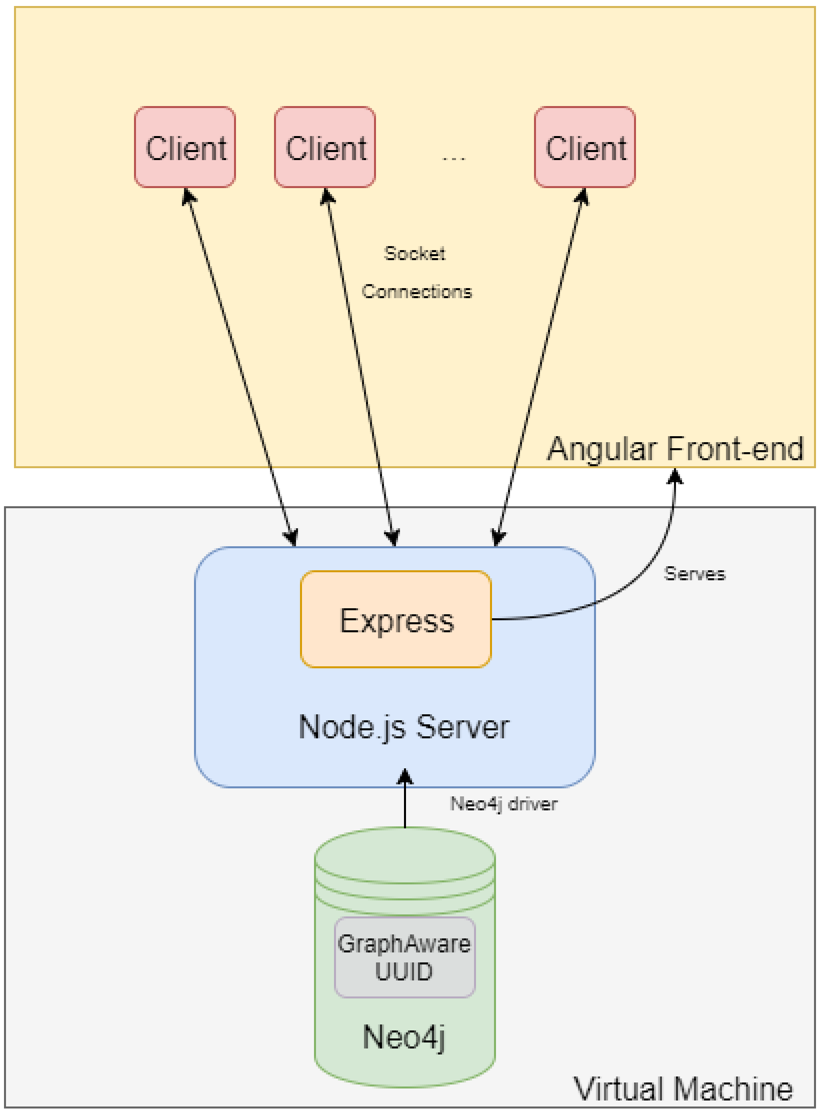

Figure 2: Detailed Software Architecture

2.3 Design Analysis

The overall design of the application using the server-client design pattern was effective. The Neo4j graph database made handling the nodes and edges of an attack graph much easier than a traditional relational database. However it needed to be extended in order to have static references to the nodes and edges.

The Node.js server was also an effective design choice. It is quite reliable and offers great throughput and rapid API responses. Using Express as a boilerplate for our web application saved us a significant amount of up-front development to get our architecture functional. It also interacts amicably with the Angular front-end.

The Angular front-end met our needs for this project but did cause several headaches. The more restrictive nature of Typescript often made object type conversions necessary and tedious. We also had some issues with serving multiple forms on the same page which led to a redesign of the user interface. But it does accomplish data-binding in a simple way and allowed for HTML templating. This made our front-end responsive and functional.

3 Implementation

This chapter strives to document the process of implementing the design as described in Chapter 2. Section 3.1 examines the project management strategies employed by the team. Section 3.2 lists several challenges we faced during the implementation process and how the team overcame or mitigated them. Section 3.3 describes the final schema of the graph database.

3.1 Project Management

Since our project is a collaborative software project, we stored all the source code online and managed it with GitLab. As a version control system, GitLab allowed us to keep track of changes and restore code to a previous version when needed. GitLab also had a nice feature for handling merge requests. As all team members worked on the project at the same time, and thus overlaps occurred, we always performed peer review together before merging our versions of the code to avoid conflicts. There was a separate branch for each developer, and then a *dev* branch for the whole team's current version and *master* branch for stable releases.

Additionally, our team used the issues board in GitLab to track team members' tasks and manage team members' workload. Any issue arising during the project was updated in the issues board for all team members to review and provide ideas to resolve. Further, we stored and collaborated on our group documents in real-time using Google Docs. This included our weekly and bi-weekly reports.

Not only collaborating online through GitLab, we also met frequently with all team members to discuss the project. We had a team meeting twice a week to address both conceptual and implementation issues. There was also weekly meeting with our client and advisor, Dr. Daniels, every

Wednesday for clarification and advice. We also conducted four “code jams”, in which we spent an entire afternoon coding and solving issues together.

3.2 Challenges

The major challenge of the project was the lack of the schema. We did not receive the python script and the attack graph schema from the graduate student as initially planned. This student’s collaboration with us was a part of the project description when we took it, but changed as the student left the project during the first semester. We have spent a considerable amount of unbudgeted time to work on how the intricacies of the attack graph relations would work.

Another issue that came up late during implementation was finding out that our initial understanding of the schema was wrong. Our plan had set up the attack graph as a sort of descending tree, with individual networks at the top, and their hosts descending from them. Upon discussion with Daniels, we found out that we neglected to consider hosts being members of multiple networks. By the time this was realized, we were too close to our deadline to change our schema.

We also faced other smaller issues during the development process. One was that the virtual machine and each team member had different Neo4j versions. Depreciations of our initial driver for Neo4j also caused unexpected change in format of data returned by the database. We, therefore, had to write an extra catch function to pipe the new input into the old output. This deprecation was caused by an update to Neo4j, where an official Javascript driver for Neo4j was created. The creator of our third-party driver decided to stop his development at that point, so we had to switch to the official version.

We discovered part of the way through design that Neo4j did not have a set identifier to reference a node or relation by, which greatly complicated how we were able to manipulate them. We resolved this issue by adding a third-party GraphAware plugin, that automatically generated an ID and attached it to every node and some types of relationships on creation.

Another issue we faced was how to handle multiple forms on the same Angular page. We redesigned the user interface to mitigate the issue. This problem was one of many that we had with Angular. None of our team had used this form of Angular that involved the use of the Typescript language, or implemented an application using Neo4j.

3.3 Schema

Since Neo4j is a graph database, it excels at storing data models based on relationships. Our schema needed to be well defined in order for it to function. The schema includes the different types of nodes (Section 3.3.1) as well as the different types of edges or relationships (Section 3.3.2). The main attack graph is comprised of Privilege nodes and Transition edges, while the other pieces within the database are extensions we have included for use within the application.

3.3.1 Nodes

The main node types stored in the database for the attack graphs are Team, Network, Host, Privilege and Flag nodes. Team nodes are used as a reference for a specific team graph as well as containing metadata about that team. The Network node is a “root” for the attack graph that it is apart of. It contains metadata about the network’s IP address and some descriptive information about the purpose of this network.

Host nodes represent a specific service or machine on the network. The final type of nodes are Privileges. These are the fundamental piece of the attack graph as they represent possible actions or permissions on a host. All other nodes are effectively used for navigating down to privileges. Flag nodes are similar to Privileges, but are distinct as they are used as endpoints for the graph traversals and are represent the attack goals of red team. Every type of node, with the exception of Flags, can be created by the users as they are discovered within the systems that the attack graph represents.

3.3.2 Relationships

There are a variety of relationships used to connect the pieces of the attack graph, and the resulting team graphs, together. The main types of relationships, or edges, are Owns, Privilege Of, Connects To, Derived From, Transition, and Hypothesis. The owns edge is used to link a Team node to a Network node. It serves as the reference between the Team and Network that is the ‘root’ of that team’s graph. The Privilege Of edge is used to connect Privilege nodes to a Host and are used for navigating the graph.

The Connects To edges are for any node that connects to a Network node, which can be other Networks or Hosts. Derived From is is used to connect a specific node in a team graph back its matching node in the scenario graph. These edges are the basis of Hypothesis creation within the application. Transition edges exist between Privileges or between a Privilege and a Flag. They are used for traversing team graphs to find paths to Flags. Transitions can be created by the users. Similar to Transition edges are the Hypothesis edges. They are created by the application whenever a new Transition is added by the user.

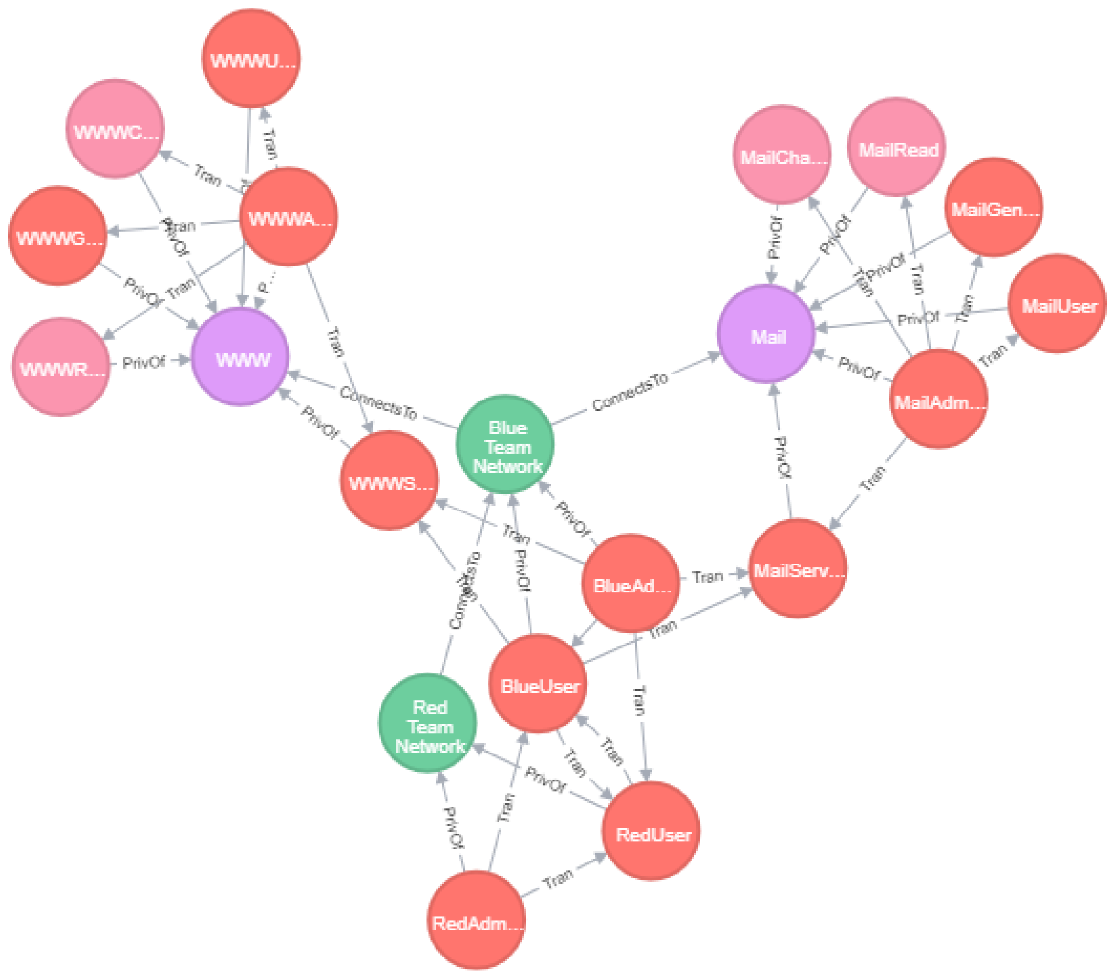

Figure 3: Example Scenario Graph

4 Testing

This chapter concerns the software testing for the project. Section 4.1 describes the testing environment used. Section 4.2 goes into more detail about the testing process for different parts of the application.

4.1 Testing Environment

For testing purposes we use Karma and Jasmine for all of our unit tests. Karma is a tool that creates an environment for the tests to be run. Jasmine is a framework that allows for a structured test environment with the help of its many functional tools. Jasmine and Karma work together to help unit tests have a structure and provide many tools to accomplish otherwise tedious and complex testing techniques.

4.2 Testing Process/Strategy

The testing process can broken down into three separate testing strategies: one for the back-end API, one for the front-end views, and one for end-to-end testing.

4.2.1 Testing API

For testing the API, ‘ngMock’ module service ‘$httpBackend’ was used to catch the http requests provided by our API services. The service is able to respond to the requests asynchronously, which is very useful for our unit tests because manually writing these tests would be more complex and harder to maintain.

4.2.2 Testing Views

Using built-in directives, such as ‘ng-show’, ‘ng-click”, we can test custom html tags, attributes, and classes. We also take advantage of an angular service called ‘$compile’, which allows us to attach specific behaviours to DOM elements or alter it completely to allow tests in a variety of environments. The services make testing complex view pages easier and maintainable.

4.2.3 E2E Testing

Some parts of our program can’t really be tested fully using manual unit tests, we use the AngularJS built-in end-to-end test runner named Protractor to run all of these tests. We are able to tell Protractor what our expectations for a certain action is, and it is able to go through the process and return if our expectations are correct or not. This tool is very helpful when having to test an action that spans across the entire program.

5 Context

This chapter serves to place our project in a greater context. Section 5.1 describes how the project could be scaled using cloud technologies from IBM, namely Bluemix. Section 5.2 is a brief research survey of academic papers that are related to our project and identifies unique aspects of our project.

5.1 Scaling with Cloud Technologies

IBM Bluemix offers functionality that would help with scaling our web application to be useable beyond the Iowa State CDC’s. Bluemix would ease deployment and maintainability with a few of its offerings. A combination of cloud hosting and auto-scaling would ease the burden of hosting and maintaining the server on Iowa State’s side as needs change for the application. As we need more storage capacity or need more processing power, Bluemix can be used to upgrade the server to add on exactly what capacities we need. This can cut down on hosting costs while the application is small.

IBM’s simple data pipe and graph databases can improve the responsiveness and uptime of the application by moving the bulk of the data onto a reliable and professionally maintained database. While our current implementation uses Neo4j for its database, replacing it with IBM Graph would potentially give us more flexibility with expansion in the future. Bluemix even has production-ready servers that match the technologies that our application is using, such as Node.js. Amazon Web Services also has template projects for a Jenkins server. This would give us a form of continuous integration to keep the application up to date with the latest set of changes.

5.2 Research Survey

Automatic generation and analysis of attack graphs:
Attack graphs can be built automatically with the NuSMV tool suite and then subsequently used to analyze a network for vulnerabilities and weaknesses. The NuSMV tool suite will produce graphs that are exhaustive and succinct. Our project automatically builds partially complete attack graphs based on a known topology, then tracks progress of an attack graphs evolution as it is manually tested by members of the red team. This “crowd-sourcing” allows our graphs to include less conventional methods of attacking a network.

A novel approach for analysis of attack graphs:
In the world of growing Internet of Things (IoT) and enterprise integrated network, vulnerabilities exist and increasingly connect to one another, creating large and complex attack graphs for which this article proposes and evaluates certain analysis framework. The framework utilized the multi-stage vulnerability analysis (MulVAL) to explore the network and identify vulnerabilities to create an attack graph. As this attack graph can potentially be complex, the article proposes a new algorithm to simplify the graph and then use the Markov model to extrapolate the risk to the entire system. Further, this framework can be applied to evaluate network security exposure and mitigate such exposure.

Using attack graphs to analyze social engineering threats:
Social engineering attacks are cheap, requires no personal experience, and can be done by anyone. Many companies suffer from these attacks every year due to negligence in taking proper precautions. These precautions include hosting security awareness trainings, incident reporting systems, and penetration testing. Going over existing access control lists periodically is also important as keeping access down to the ‘need to know’ minimizes risk. The dangers presented in the article can be represented by an attack graph. By combining the different privileges that can be gained by social engineering threats with a typical attack graph, new attack paths appear and need to be considered.

Probabilistic computing approach of attack graphs:
The article presents a method of calculating probability of every node in an attack graph, it can be used in probabilistic computing when working with large networks. The method involves using a breadth-first search in order to get a setting number, and getting a maximum and minimum probability. If the setting number lies within the max-min range, as approximate calculation is made, otherwise it’s accurate. This method could be useful to our project for certain prediction features that may be implemented in the future. By analyzing the probabilities of nodes, certain predictions can be made towards the attack graph that would be useful for finding vulnerabilities faster.

Key differences for our project:

- Crowd-sourced attack graph generation
- Handles both technical and social hacks for transitions
- Creates hypotheses for what privilege transitions “should” be there
- Prioritizes the confirmation of hypotheses based upon usefulness

Overall, although our project borrows the general concepts of attack graphs, the purpose and method are quite different. Our project uses a form of crowdsourcing to build the graphs as opposed to the usual strategy of using system analysis software. Hacktrack also treats technical and social engineering hacks similarly, and both can be added to the attack graph. Additionally, our project focuses on creating hypotheses for what privilege transitions may be in an attack graph. These hypotheses are then prioritized by their usefulness if confirmed.

6 Conclusions

In conclusion, this project was essentially a research project, as we have been unable to find any other academic sources discussing collaborative attack graph hacking. As such, this project should be seen as a proof-of-concept for the idea. We believe our project validates the concept and functions for a specific subset of CDC scenarios. However, it requires further implementation in order to handle more realistic graphs and work at scale. With regards to this, Dr. Daniels has expressed interest in having a graduate student pick up the project and continue the research or implementation.

7 References

1. S. Jha, O.Sheyner and J. Wing. “Two Formal Analyses of Attack Graphs”. http://pages.cs.wisc.edu/~jha/jha-papers/security/CSFW_2002_1.pdf. Accessed Nov 12, 2017

2. Web Design and Applications. W3C. www.w3.org/standards/webdesign

3. MEAN Stack. http://mean.io/

4. “ANNE Stack. Angular JS, Node, Neo4J and Express”. 42ID, November 5, 2014, www.42id.com/articles/anne-stack-angular-js-node-neo4j-and-express

5. IaaS Cloud Server. IBM. www.ibm.com/cloud/infrastructure

6. Auto Scaling. IBM. www.ibm.com/cloud/auto-scaling

7. Simple Data Pipe: Cloud data movement made easy. IBM. https://developer.ibm.com/clouddataservices/simple-data-pipe

8. Grap Database. IBM Cloud. https://console.bluemix.net/catalog/services/graph?env_id=ibm:yp:us-south

9. Node.js Cloud DB. IBM Cloud. https://console.bluemix.net/catalog/starters/nodejs-cloudant-db-web-starter?env_id=ibm:yp:us-south

10. Set up a Jenkins Build Server. AWS. https://aws.amazon.com/getting-started/projects/setup-jenkins-build-server/

11. Sheyner, Oleg. “Automated generation and Analysis of Attack Graphs”. IEEE Computer Society. http://www.cse.unt.edu/~rdantu/AutomatedGenerationOfAttackGraphs.pdf . Accessed Nov 12, 2017.

12. Yousefi, Mohammadmehdi. “A novel approach for analysis of attack graph”. IEEE International Conference on Intelligence and Security Informatics, 2017, ResearchOnline@GCU http://researchonline.gcu.ac.uk/portal/files/25104976/Yousefi_va2017.pdf. Accessed Nov 12, 2017

13. Beckers, Kristian. Krautsevich, Leanid. Yautsiukhin, Artsiom. “Analysis of Social Engineering Threats with Attack Graphs”. ResearchGate, September 2014. https://www.researchgate.net/profile/Artsiom_Yautsiukhin/publication/267025389_Analysis_of_Social_Engineering_Threats_with_Attack_Graphs/links/564d9bf108ae4988a7a45c41.pdf. Accessed Nov 12, 2017

14. YE Yuna, XU Xi-shana, QI Zhi-chang. “A Probabilistic Computing Approach of Attack Graph-Based Nodes in Large-scale Network”. 2011 International Conference on Environmental Science and Information Application Technology (ESIAT 2011). https://ac.els-cdn.com/S1878029611001976/1-s2.0-S1878029611001976-main.pdf?_tid=464bc5b8-c7ee-11e7-a5f4-00000aacb35e&acdnat=1510521365_00b1d90b2d875bb63ceccde1798311b2. Accessed Nov 12, 2017

Appendix I: Operations Manual

Setup

The main body of code is already on the VM along with the correct versions of the required software. In order to startup the web application, you will need to SSH onto the VM from either on campus or from behind the Iowa State VPN.

1. SSH to dec1703-hacktrack.ece.iastate.edu and sign in as user: hacktrack with password: hacktrack.
2. Become root by executing '`sudo su`' and re-entering the password: hacktrack.
3. Navigate to `/home/dpdoyle/hacktrack`.
4. Run '`git pull`' to get the latest version of the code.
5. Ensure that the shell scripts are executable by running '`chmod +x *.sh`'.

Demo

1. Run '`./startup.sh`' to start the Neo4j server and start the Node.js application.
2. The application can now be accessed from a web browser at dec1703-hacktrack.ece.iastate.edu:3000.
3. If you want to change the format of the scenario, edit the python script found at `hacktrack/database/neo4cql.py`.
4. If you want to purge the database and reload it with a new scenario, run '`./refresh_database.sh`'.
5. The application can be killed at anytime by running '`./shutdown.sh`'.

Test

Testing API and Views

1. Make sure karma.conf.js is configured to the tests
2. Run '`karma start`' to run the tests.

E2E Tests

1. Make sure the conf.js is configured to the tests
2. Run '`protractor conf.js`' to run the tests.

Appendix II: Code

The full body of code for the HackTrack project can be found at https://git.ece.iastate.edu/491_dec1703/hacktrack. Dr. Daniels (daniels@iastate.edu) is an owner of the repository, contact him for access.

Select source code files have been included below.

```javascript
const neo4j = require('neo4j-driver').v1;
const winston = require('winston');
require('express');

var db = neo4j.driver('bolt://localhost',
neo4j.auth.basic('neo4j', 'hacktrack'));

winston.info('Connected to Database: ' + JSON.stringify(db));


db.cypher = function(req, res, callback) {
    var query = req.query;
    var params = req.params;

    var session = db.session();

    session.run(query, params)
        .then(function(results) {
            if (callback) {
                callback(results);
            } else {
                var resultSet = []
                results.records.forEach(function(record) {
                    resultSet.push(record.toObject());
                })
                console.log(resultSet);
                res.status(200).json(resultSet);
            }
            session.close();
        })
        .catch(function(error) {
            winston.error(error);
            res.status(500).send(error);
            session.close();
        });
}

module.exports = db;
```

```python
#neo4cql.py

NEWLINE = "\n"
FILE = "neo4output.cql"

num_teams = 3

# Layout is [[Network],[Host1,Host2]]
sections = \
        [
            ['Red',
                ['None']
            ],
            ['Blue',
                ['Mail', 'WWW']
            ]
        ]
flags = ['Read', 'Change']
roles = ['Admin', 'User', 'Generic', 'Service']


file = open(FILE, 'w')

# Breaks alot of things
file.write("MATCH (n) WHERE NOT (n:User) DETACH DELETE n;"+NEWLINE)

# create nodes for each team
output = ""
scenarioId = ""
for i in range(num_teams + 1):
    teamId = "\"Team " + str(i) + "\""
    team = ""
    if (i == 0):
        scenarioId = "Scenario"
        team = "CREATE ("+scenarioId+":Team{name:'"+scenarioId+"', id: \""+str(i)+"\"})"+NEWLINE
    else:
        team = "CREATE (Team"+str(i)+":Team{name:"+teamId+", id: \""+str(i)+"\"})"+NEWLINE
    output += team

file.write(output)
file.write(NEWLINE)
```

```
    file.write(NEWLINE)

    # create nodes for each network
    networks = []
    userNodes = []
    nodeIds = []
    edgeIds = []
    for section in sections:
        output = ""
        internalNetUsers = []
        net = section[0];
        hosts = section[1];

        netId = net + "Net"
        nodeIds.append((netId, "Network"))
        networks.append(netId)

        network = "CREATE ("+netId+":Network{name:'"+net+" Team Network', team: \"0\"})"+NEWLINE

        adminId = net + "Admin"
        nodeIds.append((adminId, "Priv"))
        userNodes.append(adminId)
        internalNetUsers.append(adminId)
        admin = "CREATE ("+adminId+":Priv{name:'"+adminId+"', team: \"0\"})"+NEWLINE

        userId = net + "User"
        nodeIds.append((userId, "Priv"))
        userNodes.append(userId)
        internalNetUsers.append(userId)
        user = "CREATE ("+userId+":Priv{name:'"+userId+"', team: \"0\"})"+NEWLINE

        edgeIds.append((adminId, "PrivOf", netId))
        edgeIds.append((userId, "PrivOf", netId))
        netUsers = "CREATE ("+adminId+")-[:PrivOf]->("+netId+"), ("+userId+")-[:PrivOf]->("+netId+")"+NEWLINE

        output += network + admin + user + netUsers
        if (net is "Blue"):
            output += "CREATE ("+scenarioId+")-[:Owns]->("+netId+")"+NEWLINE

        file.write(output)
```

```python
        file.write(NEWLINE)

        for host in hosts:
            nodes = []
            output = ""
            if(host is "None"):
                continue

            nodeIds.append((host, "Host"))
            hostNode = "CREATE ("+host+":Host{name:'"+host+"', team: \"0\"})"+NEWLINE
            edgeIds.append((netId, "ConnectsTo", host))
            hostConnect = "CREATE ("+netId+")-[:ConnectsTo]->("+host+")"+NEWLINE
            output += hostNode + hostConnect

            for role in roles:
                roleId = host + role
                nodeIds.append((roleId, "Priv"))
                nodes.append(roleId)
                roleNode = "CREATE ("+roleId+":Priv{name:'"+roleId+"', team: \"0\"})"+NEWLINE
                output += roleNode
                if("Service" is role):
                    for user in internalNetUsers:
                        edgeIds.append((user, "Tran", roleId))
                        output += "CREATE ("+user+")-[:Tran]->("+roleId+")"

            for flag in flags:
                flagId = host + flag
                nodeIds.append((flagId, "Flag"))
                nodes.append(flagId)
                flagNode = "CREATE ("+flagId+":Flag{name:'"+flagId+"', team: \"0\"})"+NEWLINE
                output += flagNode

            output += "CREATE "
            for i in range(len(nodes)):
                node = nodes[i]
                edgeIds.append((node, "PrivOf", host))
                output+= "("+node+")-[:PrivOf]->("+host+")"
                if (i != len(nodes)-1):
                    output += ", "
            output += NEWLINE

            output += "CREATE "
            adminTag = host + "Admin"
            for i in range (len(nodes)):
                node = nodes[i]
                if "Admin" not in node:
                    edgeIds.append((adminTag, "Tran", node))
                    output += "("+adminTag+")-[:Tran]->("+node+")"
                    if (i != len(nodes)-1):
                        output += ", "
            output += NEWLINE

            file.write(output)
            file.write(NEWLINE)

    output = ""
    for i in range(len(networks)-1):
        netI = networks[i]
        for j in range(i+1, len(networks)):
            netJ = networks[j]
            edgeIds.append((netI, "ConnectsTo", netJ))
            output += "CREATE ("+netI+")-[:ConnectsTo]->("+netJ+")"+NEWLINE
    file.write(output)
    file.write(NEWLINE)

    output = ""
    for i in range(len(userNodes)):
        userA = userNodes[i]
        for j in range(len(userNodes)):
            userB = userNodes[j]
            if ((i != j) and ("Admin" not in userB)):
                edgeIds.append((userA, "Tran", userB))
                output += "CREATE ("+userA+")-[:Tran]->("+userB+")"+NEWLINE

    file.write(output)
    file.write(NEWLINE)

    for i in range(1, num_teams + 1):
        edges = []
        output = ""
        for j in range(len(nodeIds)):
            n = nodeIds[j][0]
            m = nodeIds[j][1]
            COPY = n+str(i)
            output += "CREATE ("+COPY+":"+m+") SET "+COPY+" = "+n+" SET "+COPY+".team = \""+str(i)+"\" MERGE ("+COPY+")-[:DerivedFrom]->("+n+")"+NEWLINE

            if (n == "BlueNet"):
                output += "CREATE (Team"+str(i)+")-[:Owns]->("+COPY+")"+NEWLINE

        file.write(output)
        file.write(NEWLINE)

        output = ""
        for j in range(len(edgeIds)):
            COPYa = edgeIds[j][0]+str(i)
            R = edgeIds[j][1]
            COPYb = edgeIds[j][2]+str(i)
            output += "MERGE ("+COPYa+")-[:"+R+"]->("+COPYb+")"+NEWLINE
        file.write(output)
        file.write(NEWLINE)

    file.write(";")
    file.close()
```

```
<nav class="breadcrumb" *ngIf="teamId && networkId">
    <a class="breadcrumb-item" routerLink="/">Home</a>
    <a class="breadcrumb-item" routerLink="/teams">Teams</a>
    <a class="breadcrumb-item" routerLink="/teams/{{teamId}}">Team</a>
    <a class="breadcrumb-item" routerLink="/network/{{networkId}}">Network</a>
    <a class="breadcrumb-item" *ngIf="hostId" routerLink="/host/{{hostId}}">Host</a>
    <span class="breadcrumb-item active">Privilege</span>
</nav>
<h2>{{privilege.name}}</h2>

<h3>In</h3>
<div *ngFor="let tran of in">
    <a routerLink="/trans/{{tran.uuid}}">#{{tran.uuid}}</a> -> {{tran.start.name}}
</div>
<div *ngFor="let hypo of inHypo">
    <a routerLink="/hypo/{{hypo.uuid}}"><i>#{{hypo.uuid}}</i></a> -> <i>{{hypo.start.name}}</i> [Hypothesis]
</div>
<p *ngIf="!in.length"><i>No inbound transistions</i></p>

<h3>Out</h3>
<div *ngFor="let tran of out">
    <a routerLink="/trans/{{tran.uuid}}">#{{tran.uuid}}</a> -> {{tran.end.name}}
</div>
<div *ngFor="let hypo of outHypo">
    <a routerLink="/hypo/{{hypo.uuid}}"><i>#{{hypo.uuid}}</i></a> -> <i>{{hypo.start.name}}</i> [Hypothesis]
</div>
<p *ngIf="!out.length"><i>No outbound transistions</i></p>

<button class="btn btn-primary" (click)="showForm()">New Transition</button>
<div *ngIf="newTransition">
    <transition-form></transition-form>
    <br>
    <button class="btn btn-default" (click)="hideForm()">Hide</button>
</div>
```

```
    </div>
```

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute, Params } from '@angular/router';
import { Privilege, Transition, Hypothesis } from '../../_models/index';
import { PrivilegeService, TransitionService, HypothesisService } from '../../_services/index';

import 'rxjs/add/operator/switchMap';

@Component({
    templateUrl: 'privilege.component.html',
})
export class PrivilegeComponent implements OnInit {
    privilegeId: string;
    hostId: string;
    networkId: string;
    teamId: string;
    privilege: Privilege = {
        uuid: null,
        name: null,
        team: null
    };
    in: Transition[] = [];
    out: Transition[] = [];
    inHypo: Hypothesis[] = [];
    outHypo: Hypothesis[] = [];

    newTransition = false;

    constructor(
        private privilegeService: PrivilegeService,
        private transitionService: TransitionService,
        private hypothesisService: HypothesisService,
        private route: ActivatedRoute) {}

    ngOnInit() {
        this.route.params.subscribe((params: Params) => {
            this.privilegeId = params['id'];
            this.loadPrivilege();
        });
    }

    showForm(){
        this.newTransition = true;
```

```
            this.newTransition = true;
        }

        hideForm(){
            this.newTransition = false;
        }

        private loadPrivilege() {
            this.privilegeService.getById(this.privilegeId).subscribe(priv => {
                this.privilege = priv;
                this.getContext(priv);
                this.transitionService.getInbound(this.privilege).subscribe(inbound => { this.in = inbound; });
                this.transitionService.getOutbound(this.privilege).subscribe(outbound => { this.out = outbound; });
                this.hypothesisService.getInbound(this.privilege).subscribe(inbound => {this.inHypo = inbound; });
                this.hypothesisService.getOutbound(this.privilege).subscribe(outbound => {this.outHypo = outbound; });
             });
        }

        private getContext(privilege){
            var hostCtx, networkCtx;
            this.privilegeService.networkContext(privilege).subscribe(res => {
                networkCtx = res;
                this.privilegeService.hostContext(privilege).subscribe(res => {
                    hostCtx = res;
                    console.log('nCtx', networkCtx);
                    console.log('hCtx', hostCtx);
                    if(networkCtx[0]){
                        var resp = networkCtx[0];
                        this.teamId = resp.team.properties.id;
                        this.networkId = resp.network.properties.uuid;
                        this.hostId = null;
                    }
                    else {
                        var resp = hostCtx[0];
                        this.teamId = resp.team.properties.id;
                        this.networkId = resp.network.properties.uuid;
                    this.hostId = resp.host.properties.uuid;
                }
                console.log(this);
            });
        });

    }

}
```

```
// Get dependencies
const express = require('express');
const path = require('path');
const http = require('http');
const bodyParser = require('body-parser');
const winston = require('winston')

//Setup Winston for logging
//transport for all messages
winston.add(winston.transports.File, {
    name: 'out-log',
    filename: './logs/out.log',
    json: false
});
//transport for error messages
winston.add(winston.transports.File, {
    name: 'error-log',
    filename: './logs/error.log',
    level: 'error',
    json: false
});

// Get our API routes
const test = require('./server/routes/api');

const users = require('./server/routes/users');
const teams = require('./server/routes/teams');
const tasks = require('./server/routes/tasks');
const traversals = require('./server/routes/traversals');
const networks = require('./server/routes/networks');
const bulletins = require('./server/routes/bulletins');
const hosts = require('./server/routes/hosts');
const privileges = require('./server/routes/privileges');
const flags = require('./server/routes/flags');
const trans = require('./server/routes/trans');
const hypo = require('./server/routes/hypo');

const app = express();

// Parsers for POST data
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: false
}));
```

```
// Point static path to public
app.use(express.static(path.join(__dirname, 'public')));

// Set our api routes
app.use('/api/test', test);
app.use('/api/users', users);
app.use('/api/tasks', tasks);
app.use('/api/teams', teams);
app.use('/api/traversals', traversals);
app.use('/api/bulletins', bulletins);
app.use('/api/networks', networks);
app.use('/api/hosts', hosts);
app.use('/api/privileges', privileges);
app.use('/api/flags', flags);
app.use('/api/trans', trans);
app.use('/api/hypo', hypo);

// Catch all other routes and return the index file
app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, 'public/index.html'));
});

/**
 * Get port from environment and store in Express.
 */
const port = process.env.PORT || '3000';
app.set('port', port);

/**
 * Create HTTP server.
 */
const server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */
server.listen(port, "0.0.0.0", () => winston.log('info', 'API
running on localhost:%d', port));
```

```javascript
const express = require('express');
const router = express.Router();
const winston = require('winston');
const db = require('../db');

router.get('/', (req, res) => {
    db.cypher({
        query: 'MATCH ()-[tran:Tran]->() RETURN tran;',
        params: {}
    }, res);
});

//get a specific transistion
router.get('/:id', (req, res) => {
    winston.info("get trans %s", req.params.id)
    db.cypher({
        query: 'MATCH (start)-[tran:Tran]->(end) WHERE
        tran.uuid={id} RETURN tran, start, end;',
        params: {
            id: req.params.id
        }
    }, res);
});

//get all inbound transistions to a given privilege
router.get('/priv/:id/in', (req, res) => {
    db.cypher({
        query: 'MATCH (p:Priv)<-[tran:Tran]-(start) WHERE
        p.uuid={priv} RETURN tran, start;',
        params: {
            priv: req.params.id
        }
    }, res);
});

//get all outbound transistions to a given privilege
router.get('/priv/:id/out', (req, res) => {
    db.cypher({
        query: 'MATCH (p:Priv)-[tran:Tran]->(end) WHERE
        p.uuid={priv} RETURN tran, end;',
        params: {
            priv: req.params.id
        }
    }, res);
});
```

```
});

//create a new transistion
router.post('/', (req, res) => {
    var data = req.body;
    winston.info("create new transistion");
    db.cypher({
        query: 'MATCH (start:Priv), (end:Priv) ' +
            'WHERE start.uuid={start} AND end.uuid={end} ' +
            'CREATE (start)-[tran:Tran' +
            '{foundBy:{finder},description:{desc},howTo:{howTo}
            ,team:{team}}' +
            ']->(end) RETURN tran;',
        params: {
            start: data.fromPrivilege,
            end: data.toPrivilege,
            finder: data.foundBy,
            desc: data.description,
            howTo: data.howTo,
            team: data.team
        }
    }, res, DetectHypothesisTargets(res, data));

});

//On successful creation of path, create hypothesis' on team
graphs that have the same nodes and are shown on the scenario
graph
function DetectHypothesisTargets(res, data){
    //Indicate successful transition creation, res won't be
    used in the rest of this callback chain
    res.status(200);
    //Check if there is a complete transition path to a flag
    pathCheck(res, data.team);
    db.cypher({
        query: "MATCH (:Priv {uuid:
        {start}})-[:DerivedFrom]->(ScenarioStart:Priv)<-[:Deriv
        edFrom]-(TeamStarts:Priv) WHERE NOT TeamStarts.team =~
        {teamNum} MATCH (:Priv {uuid:
        {end}})-[:DerivedFrom]->(ScenarioEnd:Priv)<-[:DerivedFr
        om]-(TeamEnds:Priv) WHERE NOT TeamEnds.team =~
        {teamNum} AND TeamStarts.team =~ TeamEnds.team  RETURN
        TeamStarts, TeamEnds",
        params: {
            teamNum: data.team,
            start: data.fromPrivilege,
            end: data.toPrivilege,
        }
    }, res, function(results){
        results.records.forEach(function(element){
            var start = element._fields[0].properties.uuid;
            var team = element._fields[0].properties.team;
            var end = element._fields[1].properties.uuid;

            createHypothesis(res, start, end, data);
        });
    });
}

//Create a hypothesis between the given start and end node on
a single team
function createHypothesis(res, start, end, data) {
    db.cypher({
        query: "MATCH (from:Priv { uuid: {start}}), (to:Priv
        {uuid: {end}}) CREATE (from)-[r:Hypo
        {foundBy:{finder},description:{desc},howTo:{howTo}}]->(
        to) RETURN r",
        params: {
            start: start,
            end: end,
            finder: data.foundBy,
            desc: data.description,
            howTo: data.howTo,
        }
    }, res, function(results){
        hypoID = results.records[0]._fields[0].identity.low;
        createTask(res, data.team, hypoID, null);
    });
}

//Create a Task that points to a hypothesis for confirming or
a flag for capture
//TODO priorities?
function createTask(res, team, hypoID, flagID){
    console.log("Hypo:" + hypoID);
    console.log("Flag:" + flagID);
    if(hypoID){
        db.cypher({
        query: "MATCH (team:Team) WHERE team.id = {team} MATCH
        ()-[s]->() WHERE ID(s) = toInteger({hypo}) MERGE
        (task:Task {priority: '1', hypothesis: s.uuid,
        complete: False})-[r:AssignedTo]->(team) RETURN task",
        params: {
            hypo: hypoID,
            team: team,
        }
    }, res, function(){});
    } else if (flagID) {
        db.cypher({
        query: "MATCH (team:Team) WHERE team.id = {team} MERGE
        (task:Task {priority: '1', flag: {flag}, complete:
        False})-[r:AssignedTo]->(team) RETURN task",
        params: {
            flag: flagID,
            team: team,
        }
    }, res, function(){});
    }
};

function pathCheck(res, team){
    //TODO create tasks only for unclaimed flags
    db.cypher({
        query: "MATCH (n:Priv { name: 'RedAdmin', team:
        {team}}), p = shortestPath((n)-[r:Tran*1..]->(f:Flag
        {team: {team}})) RETURN p",
        params: {
            team: team,
        }
    }, res, function(results){
        console.log("PATH CHECK:");
        console.log(results);
        if(results){
            createTask(res, team, null, '10')
            //TODO add in reference to actual flag(s)
        }
    });
}

module.exports = router;
```

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Transition, Privilege } from '../_models/index';

@Injectable()
export class TransitionService {
    constructor(private http: Http) { }

    getAll() {
        return this.http.get('/api/trans').map((response:
        Response) => {
                var res = response.json();
                let trans : any[] = [];
                for(var r in res){
                    trans[r] = res[r].trans.properties;
                }
                return trans;
        });
    }

    getById(id: String){
        return this.http.get('/api/trans/' + id).map((response:
        Response) => {
                var res = response.json()[0];
                var tran = res.tran.properties;
                tran.fromPrivilege = "" +
                res.start.properties.uuid;
                tran.fromType = res.start.labels[0];
                tran.toPrivilege = "" + res.end.properties.uuid;
                tran.toType = res.end.labels[0];
                return tran;
        });
    }

    getInbound(priv: Privilege) {
        return this.http.get('/api/trans/priv/' + priv.uuid +
        '/in').map((response: Response) => {
                var res = response.json();
                let trans : any[] = [];
                for(var r in res){
                    trans[r] = res[r].tran.properties;
                    trans[r].start = res[r].start.properties;
                }
                return trans;
        });
        });
    }

    getOutbound(priv: Privilege) {
        return this.http.get('/api/trans/priv/' + priv.uuid +
        '/out').map((response: Response) => {
                var res = response.json();
                let trans : any[] = [];
                for(var r in res){
                    trans[r] = res[r].tran.properties;
                    trans[r].end = res[r].end.properties;
                }
                return trans;
        });
    }


    create(transition: Transition){
        return this.http.post('/api/trans/',
        transition).map((response: Response) =>
        (response.json())[0].tran.properties);
    }

}
```