CySec-Game

Final Report

sdmay21-50

Client: Manimaran Govindarasu

Team Members: Harrison Majerus,

Nicholas Battani, Hayden Sellars,

Jonathan Greazel, Joseph Strobel, Stefan Peng

Website: https://sdmay21-50.sd.ece.iastate.edu/

04/25/2021

Executive Summary

Development Standards & Practices Used

- Agile development
- Common Vulnerabilities and Exposures (CVE)
- Common Vulnerability Scoring System (CVSS)
- National Vulnerability Database (NVD)

Summary of Requirements

- The software shall be remotely accessible through a web application
- The software shall run use-cases of risk assessment scenarios using PowerCyber test bed
- The software shall be integrated with existing testbed and simulation tools
- The software shall utilize Game Theory algorithms for risk assessment of cyber physical systems like the smart grid and provide best strategies to mitigate the risk
- The software's UI design shall encourage ease of use and work to minimize clicks per action
- Different environments (Windows, Linux, etc.) shall have no effect on software usability
- The software shall cost no more than $0
- A working proof of concept and completed documentation shall be delivered by the end of Spring 2021 semester
- Any sensitive user information shall be stored in a safe and secure manner
- All work shall be original for our development team with credit given to proper sources

Applicable Courses from Iowa State University Curriculum

- Com S 228
- Com S 309
- Com S 319
- Math 314
- CPRE 530
- SE 329
- SE 339

New Skills/Knowledge acquired that was not taught in courses

- Game theory
- Cybersecurity for critical infrastructure
- Network modeling

Table of Contents

Executive Summary

Table of Contents

1 Project Design
- 1.1 Acknowledgement
- 1.2 Problem and Project Statement
- 1.3 Design Evolution
- 1.4 Requirements
  - Functional Requirements
  - Non-Functional Requirements
  - Analysis-Specific Requirements

2 Implementation Details
- 2.1 Implementation Methodology
- 2.2 Client Side Implementation
- 2.3 Server Side Implementation
- 2.4 Analysis algorithms
  - Attack tree engine
  - Attack defense tree engine
  - Game theory engine
- 2.5 Constraints

3 Testing
- 3.1 Unit Testing
- 3.2 Interface Testing
- 3.3 Acceptance Testing
- 3.4 Results
- 3.5 Case Study

4 Related Works

5 Conclusion

6 References

Appendix I - Operation Manual
- I.1 Setup
- I.2 App Layout
  - I.2.1 Palette
  - I.2.2 Diagram Area
  - I.2.3 Diagram Variables
  - I.2.4 Analysis Engine Selector
  - I.2.4 Analyze Button
- I.3 Creating a Diagram
  - I.3.1 Attack Tree
  - I.3.2 Attack Defense Tree

I.3.3 Game Theory
I.4 Results
I.4.1 Attack Tree
I.4.2 Attack Defense Tree
I.4.3 Game Theory

Appendix II - Other Considerations

1 Project Design

1.1 Acknowledgement

We would like to thank our project advisor/client Dr. Manimaran Govindarasu and graduate students Burhan Hyder and Kush Khanna for assisting us in our understanding and design of this project.

1.2 Problem and Project Statement

Critical infrastructure like power and energy systems are often vulnerable to cyber attacks. Mitigating cyber risk to critical infrastructure is an important part of the design and maintenance of these systems. These power and energy systems commonly use legacy devices where complete upgrades are uneconomical, therefore, risk assessment plays an important role in selectively securing vulnerable or high-risk assets.

The goal of this project is to develop a software tool that helps the critical infrastructure industry to assess cyber risks to power and energy systems and to optimally allocate cybersecurity investments to mitigate the risks. This tool will use attack tree, attack-defense tree, and game theory models to identify high-risk targets and suggest investments to mitigate the identified risks.

1.3 Design Evolution

Since the end of CPRE 491, our design has undergone some evolution as we have communicated with the client and gained a greater understanding of the project and the project's requirements.

On the client side, we have removed the need for a login page as the app will not store user data. In turn it is on the user to recreate their attack tree each time they use the system. In the future, functionality may be implemented for the user to be able to login and store trees for later use. The client side has also removed the need to animate attacks while waiting on the backend as this is a much shorter process than envisioned. As far as overall UI goes, we opted for a single page application instead of having separate pages for input and output. This design allows for the user to continue to see their tree alongside the output from the analysis. In the future, the user will be able to click on attack scenarios and they will illuminate in the tree so that users can see exactly where they are most likely to be compromised and make defense investments.

The method of backend analysis changed a few times over the course of the Spring semester. Our original design had three analysis engines, but we changed to two engines (dropping attack-defense tree analysis and leaving attack tree and game theory) as we discussed the feasibility of completing three in the semester; later opting for the third as it became more realistic to accomplish.

We also changed specific details about how the analysis should be calculated in the attack tree and attack-defense tree algorithms. Some changes included normalizing probabilities instead of forcing the user to enter exact probabilities, changing to a risk percentage for the attack-defense analysis as we concluded it was easier to interpret, and limiting the risk recalculation when defending so the risk percentage of a successful attack scenario wouldn't increase if it did not have defended attack.

1.4 Requirements

Functional Requirements

- The software shall be remotely accessible through a web application
- The software shall utilize Game Theory for risk assessment of cyber physical systems and provide best strategies to mitigate the risk
- The software's UI design shall encourage ease of use and work to minimize clicks per action

- Run use-cases of risk assessment scenarios with the tool
- Run use-cases of cybersecurity investment scenarios using the tool

Non-Functional Requirements

- Different environments (Windows, Linux, etc.) shall have no effect on software usability
- The software shall cost no more than $0
- Any sensitive user information shall be stored in a safe and secure manner

Analysis-Specific Requirements

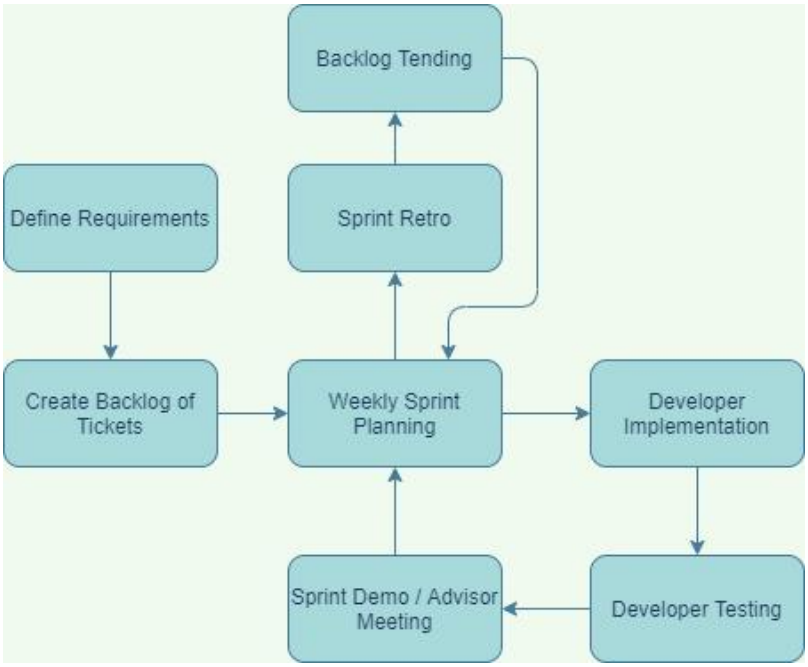
- Attack Tree
    - Risk threshold is given for tree (acceptable level of risk)
    - Each attack node has an associated attack probability
    - Root node (of the tree) has the attack impact value
    - Attack paths are enumerated (with path probability)
    - Risk index is calculated for each attack path as Risk(path) = prob(path) * impact
    - List all the paths in decreasing order of risk index
- Attack-Defense Tree
    - Risk threshold is given for tree (acceptable level of risk)
    - Each attack node has a defense mechanism with a cost associated with it
    - Each attack node has two probabilities – one denoting “pre-defense” probability, the other denoting “post-defense” probability and post-defense_prob(attack) <= pre-defense_prob(attack)
    - There is a fixed budget provided for security investment
    - Mitigate the high risks paths (risks > acceptable threshold), cybersecurity investment analysis needs to be done (by activating defense measures associated with the leaves)
- Game theory
    - Each attack node has an associated attack probability and attack cost
    - Each attack node has a corresponding defense node with a defense cost
    - Successful attack and defense scenarios and their corresponding costs are calculated
    - The payoff for each combination of attack and defense scenario is calculated
    - The Nash equilibrium is generated from the payoff matrix
    - The recommended investment strategy is determined from the Nash equilibrium and the defense budget

2 Implementation Details

2.1 Implementation Methodology

Implementation began during the first semester with rigorous requirement definition. The mathematical and system abstraction concepts can be difficult to comprehend without prior knowledge so there were many discussions on how to best serve users. The team eventually settled on a diagram based design where users can create a system diagram and manually load values. The analyzed results would then be presented along the right side of the screen.

Throughout the semester our team followed an agile-scrum schedule where weekly advisor meetings served as spring demonstrations that were followed by sprint planning and a week long sprint. At irregular intervals we would spend time during our planning meetings to take a retrospective look at what work we'd done so far and how to improve our workflow. Tending to our backlog was also a semi-regular affair, since our advisor was able to make requirement changes and requests during demonstrations.



Our project focused only on local development since the scope of the project was for a proof-of-concept application that could be expanded upon in the future. However we did build out a back end API to reduce coupling and future proof the software for potentially adding a database or remote hosting.

2.2 Client Side Implementation

On the client side we utilized Angular and TypeScript to create a modern, performant, component based application; we decided Angular was the best choice for this project based on how easily we could get the project set up and running without taking the time to customize build tooling or configuring TypeScript features. While Angular provides the foundation for the project, the main functionality is provided by go.js, a library for building interactive

diagrams and graphs. And finally, for UI components we settled on Ant Design, their library was the most robust despite it having a slightly larger build time than some alternatives.

2.3 Server Side Implementation

Our server side code hosts all of our heavy mathematical logic and analysis engines. We classified these algorithms as business logic and didn't want them easily accessible from the browser. This logic was made easier by the use of Nash.py, a library used for the computation of equilibria in 2 player strategic form games written in Python. The availability of this library made the use of Python and Django for the API an easy decision.

2.4 Analysis algorithms

Each analysis engine normalizes input probabilities such that the probability of attack nodes sum to 1. After normalization, each engine computes the relevant attack and defense scenarios, performs further calculations, and returns a result to the front end.

In the following sections, an "attack scenario" is a combination of attack nodes that leads to a compromise of the root node. The probability of an attack scenario is the product of each node in the scenario, while the cost is the sum of the weighted costs of each node in the scenario.

Attack tree engine

This engine performs a recursive depth-first search of the input network tree to find all possible attack scenarios. The core logic is given below.

- If the node is a leaf node, the only possible attack scenario is the set containing the leaf node
- If the node is an AND node, the set of possible attack scenarios is the Cartesian product of the child nodes' attack scenario sets
- If the node is an OR node, the set of possible attack scenarios is given by the union of the child nodes' attack scenario sets

Attack defense tree engine

This engine computes the set of successful attack scenarios in a similar manner to the attack-tree engine. The addition is that the algorithm also will determine a good defense strategy given the cost of deploying a defense against each attack, how effective each defense is (change in probability), the allowed defense budget, and the acceptable risk threshold (percentage risk).

The algorithm will loop through successful attack scenarios and if the risk percentage of the scenario is higher than the acceptable risk threshold, it will look through all defenses that could apply to that scenario and will defend an attack with the highest return on investment (probability/cost) if it is within budget. It will keep on defending attacks until none of them of the defenses are within the current budget or the scenario's risk percentage drops below the acceptable level.

The output includes the successful attack scenarios along with their pre-defense and post-defense risk percentage, the attacks defenses should be deployed against, and the cost of defense.

Game theory engine

Like the attack tree engine, the game theory engine also performs a recursive depth-first search of the input network tree, but it computes defense scenarios in addition to attack scenarios. Here, a "defense scenario" is a combination of defense nodes that prevents a compromise of the root node (used in the game theory engine).

The tree recursion is called on the root node of the diagram and works as follows:

- If the node is a leaf node
    - The only set of attack scenarios is the set containing the leaf node

    - The only set of defense scenarios is the set containing the corresponding defense node
- If the node is an AND node
    - The set of possible attack scenarios is the Cartesian product of the child nodes' attack scenario sets
    - The set of defense scenarios is the union of the child nodes' defense scenario sets
- If the node is an OR node
    - The set of possible attack scenarios is given by the union of the child nodes' attack scenario sets
    - The set of possible defense scenarios is the Cartesian product of the child nodes' defense scenario sets

After computing the set of attack and defense scenarios, the payoff matrix for the system is calculated. For each combination of attack and defense scenarios, the payoff is given by $U_a = C_d + I - C_a$, where U_a is the attacker's payoff, C_d is the cost of the defense scenario, I is the impact to the system, and C_a is the cost of the attack scenario. The corresponding payoff for the defender is given by $U_d = -U_a$ as the system is treated as a zero-sum game. This results in a matrix similar to the one given below, with attacker payoffs in red and defender payoffs in green.

Attacker/Defender	Defense scenario 1	Defense scenario 2	Defense scenario 3	Defense scenario 4
Attack scenario 1	4.8,-4.8	8.8,-8.8	5.8,-5.8	9.8,-9.8
Attack scenario 2	6.9,-6.9	10.9,-10.9	7.9,-7.9	11.9,-11.9

We then use the Nash.py library to compute the Nash equilibria from the payoff matrix. This tells us the probability with which the attacker and defender should choose each strategy. In the example above, the highlighted combination, Attack scenario 2 and Defense scenario 1, was suggested with a probability of 1. This means that the attacker should always choose Attack scenario 2 while the defender should always choose Defense scenario 2.

We then proportionally assign the defense budget to each defense scenario in the Nash equilibria based on its probability. Within each scenario, the budget is split amongst the defense nodes in a similar manner based on their defense costs. The list of nodes in each defense scenario contained in the Nash equilibria, along with their associated recommended investment, is then returned to the frontend to be displayed to the user.

2.5 Constraints

One major constraint of the CySec tool is a reliance on quality data from the user. When a user is inputting data they are expected to provide probabilities that a given attack and defense may happen or be defended. It is the responsibility of the user to provide as accurate of a probability as they can, however it is not always clear how likely it is for an attack to happen or a defense to be readily prepared to defend the attack. Thus, the tool will process the given data and return an optimal investment strategy, but this investment may not be the true optimization if the user inaccurately estimates probabilities.

A second constraint to the CySec tool is the current implementation relies on a paid graphing library on the frontend. This library allows the user to easily input an attack tree and provides ease to the developers because they do not have to write their own implementation of graphing. There is no current budget available for this tool and thus it would not be able to be used in a production environment without proper licensing fees paid to the owner of the graphing library.

3 Testing

Testing for this project was broken up into frontend, backend, and API testing. Frontend testing was done through unit tests written in Angular, as well as manual testing performed by frontend developers and the project advisor. Backend testing was done through unit tests written in Python on top of full engine tests that were performed with predetermined input and comparing it to expected output. For the API, testing was performed using the Postman application to ensure data is sent correctly in both directions.

3.1 Unit Testing

To test the frontend components and framework, the Jasmine unit testing framework was used to ensure all parts were working as intended. The backend was unit tested using the python unittest framework. Both the frontend and backend unit tests were used to ensure basic functionality and reduce errors caused by updates to the codebase.

3.2 Interface Testing

Interface testing was performed on the API using the Postman application as well as manual testing. The API was tested in Postman to ensure correct data was being sent and received from the frontend to the backend.

3.3 Acceptance Testing

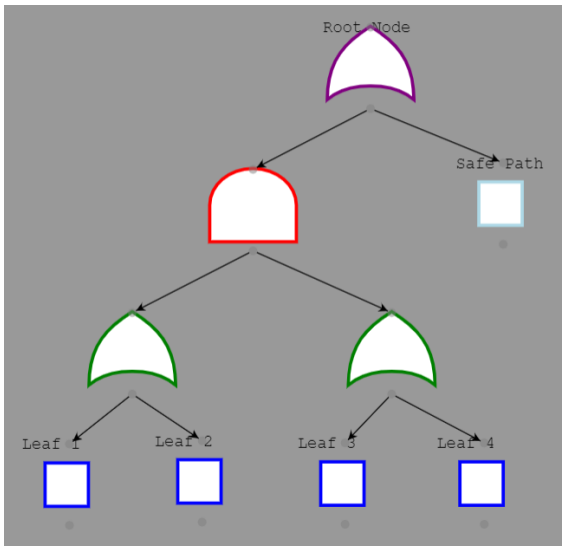
Acceptance testing was performed on a weekly basis with the project advisor and client. A demo of the progress made in the past week's sprint was given, and discussions on updates to be made occurred. These weekly demos allowed for our project to adjust to the clients requirements and needs in real time.

3.4 Results

Attack tree results:
Input:

Impact: 1000

Node	Probability Value
Safe Path	5
Leaf 1	8
Leaf 2	4
Leaf 3	6
Leaf 4	7



Expected analysis:

First normalization of inputs into probabilities
5+8+4+6+7 = 30

Node	Probability Value	Normalized Probability
Safe Path	5	5/30 = 0.1666
Leaf 1	8	8/30 = 0.2666
Leaf 2	4	4/30 = 0.1333
Leaf 3	6	6/30 = 0.2000
Leaf 4	7	7/30 = 0.2333

Next, find all scenarios of attack that would compromise the root node. Since this example is simple, just by a visual analysis of the tree we can elicit the following scenarios: (‘scenario’, when used below really mean ‘successful attack scenario’)

Scenario	Attacks
Scenario 1	Leaf 1, Leaf 3
Scenario 2	Leaf 1, Leaf 4
Scenario 3	Leaf 2, Leaf 3
Scenario 4	Leaf 2, Leaf 4

By multiplying the probability of each leaf present in a successful attack scenario, we can find the probability of that scenario occurring

Scenario	Probability
Scenario 1	0.2666 * 0.2000 = 0.0533
Scenario 2	0.2666 * 0.2333 = 0.0622
Scenario 3	0.1333 * 0.2000 = 0.0267
Scenario 4	0.1333 * 0.2333 = 0.0311

And risk is calculated as probability * impact

Scenario	Impact
Scenario 1	53.3
Scenario 2	62.2
Scenario 3	26.7
Scenario 4	31.1

Now we have all the information we want

Scenario	Attacks	Probability	Impact
Scenario 1	Leaf 1, Leaf 3	0.0533	53.3
Scenario 2	Leaf 1, Leaf 4	0.0622	62.2
Scenario 3	Leaf 2, Leaf 3	0.0267	26.7
Scenario 4	Leaf 2, Leaf 4	0.0311	31.1

Actual Output:

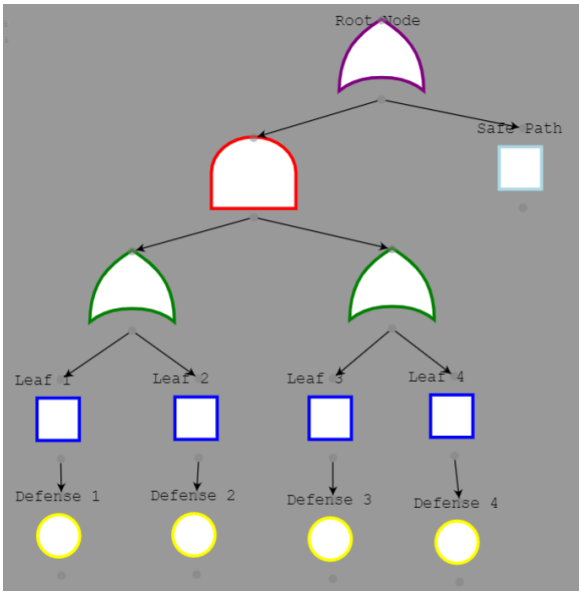
[{"risk": 53.3333, "probability": 0.0533, "leafKeys": ["LEAF", "LEAF3"]},
{"risk": 62.2222, "probability": 0.0622, "leafKeys": ["LEAF", "LEAF4"]},
{"risk": 26.6667, "probability": 0.0267, "leafKeys": ["LEAF2", "LEAF3"]},
{"risk": 31.1111, "probability": 0.0311, "leafKeys": ["LEAF2", "LEAF4"]}]

The expected analysis matches the actual output

Attack-Defense tree results:
Input:

Acceptable Risk Threshold: 18%
Budget: 300

Node	Pre-Defense Probability Value	Post-Defense Probability Value	Cost to Defend
Safe Path	5		
Leaf 1	8	5	250
Leaf 2	4	3	150
Leaf 3	6	3	200
Leaf 4	7	5	100



Expected Analysis:

First normalization of inputs into probabilities
5+8+4+6+7 = 30
Note that post-defense is normalized to the pre-defense sum in this implementation

Node	Probability Value	Normalized Pre-Defense Probability	Normalized Post-Defense Probability
Safe Path	5	5/30 = 0.1666	
Leaf 1	8	8/30 = 0.2666	5/30 = 0.1666
Leaf 2	4	4/30 = 0.1333	3/30 = 0.1000
Leaf 3	6	6/30 = 0.2000	3/30 = 0.1000
Leaf 4	7	7/30 = 0.2333	5/30 = 0.1666

Next, find all scenarios of attack that would compromise the root node

Scenario	Attacks
Scenario 1	Leaf 1, Leaf 3
Scenario 2	Leaf 1, Leaf 4
Scenario 3	Leaf 2, Leaf 3
Scenario 4	Leaf 2, Leaf 4

By multiplying the probability of each leaf present in a successful attack scenario, we can find the probability of that scenario occurring

Scenario	Pre-Defense Probability
Safe Path	0.1666
Scenario 1	0.0533
Scenario 2	0.0622
Scenario 3	0.0267
Scenario 4	0.0311

Now successful attack scenarios are normalized with the safe path to create a risk percentage:
0.1666+0.0533+0.0622+0.0267+0.0311 = 0.3399

Scenario	Risk
Safe Path	0.4901
Scenario 1	0.1568
Scenario 2	0.1830
Scenario 3	0.0786
Scenario 4	0.0915

Now we will iterate through the scenarios by percentage risk and find best options to defend those which are above the acceptable risk threshold.

Scenario 2 has a higher percentage risk than allowed, so we look into investment option by best return on investment (ROI).

Scenario 2 contains Leaf 1 and Leaf 4, ROI is determined by probability change vs cost of defense.

Attack	ROI eq	ROI
Leaf 1	(8-5)/250	0.012
Leaf 4	(7-5)/100	0.02

Leaf 4 has the greater ROI, so since it is within the current budget left we add it to our list of defended attacks.

Budget is now 300-100 = 200.

Recalculate risk percentages for all affected scenarios.

Scenario	Risk	New Risk
Safe Path	0.4901	0.4901
Scenario 1	0.1568	0.1568
Scenario 2	0.1830	0.2666 * 0.1666 / 0.3399 = 0.1307
Scenario 3	0.0786	0.0786
Scenario 4	0.0915	0.1333 * 0.1666 / 0.3399 = 0.0653

Now all the scenarios are below the acceptable threshold
Summary
Cost : 100 out of available 300
Invested nodes: Leaf 4

Scenario	Attacks	Pre-Defense Risk	Post-Defense Risk
Scenario 1	Leaf 1, Leaf 3	0.1568	0.1568
Scenario 2	Leaf 1, Leaf 4	0.1830	0.1307
Scenario 3	Leaf 2, Leaf 3	0.0786	0.0786
Scenario 4	Leaf 2, Leaf 4	0.0915	0.0653

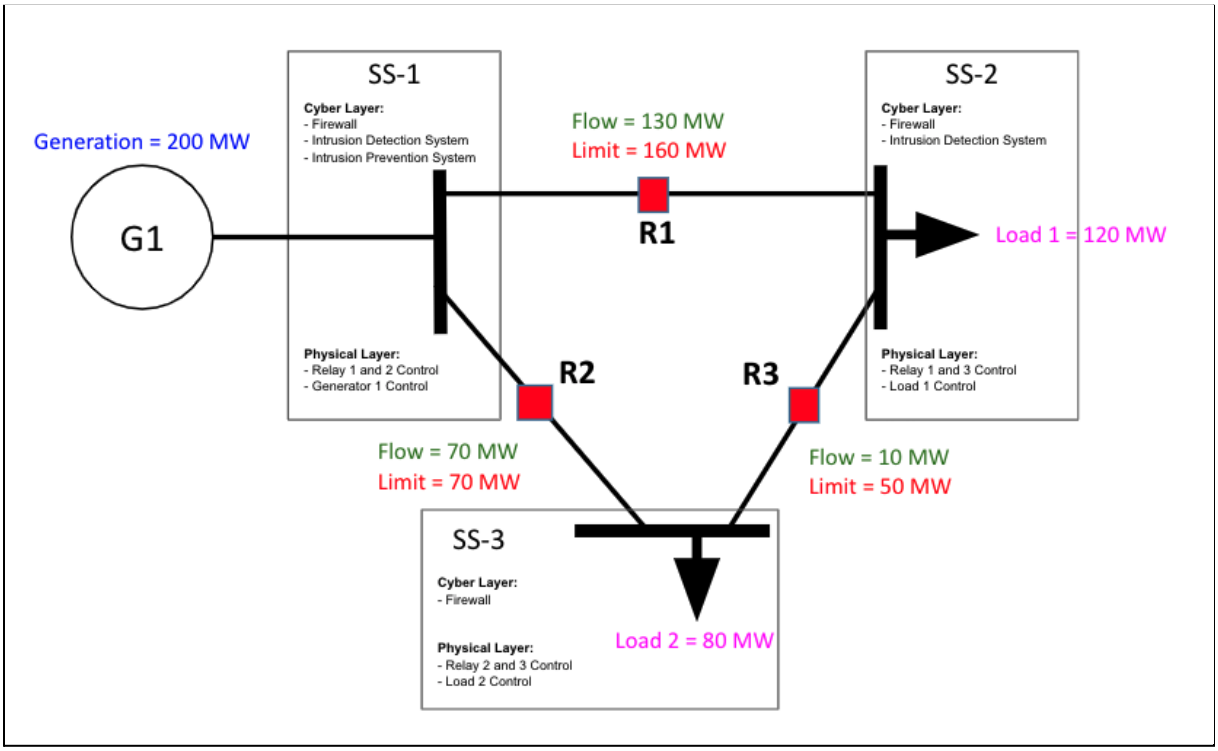
Actual output:

{"cost": 100, "defendedAttacks": ["LEAF4"], "attackScenarios":
[{"risk": 0.15686274509803924, "preDefenseRisk": 0.15686274509803924, "attacks":
[["LEAF", "Leaf 1"], ["LEAF3", "Leaf 3"]]},
{"risk": 0.13071895424836602, "preDefenseRisk": 0.18300653594771243, "attacks":
[["LEAF", "Leaf 1"], ["LEAF4", "Leaf 4"]]},
{"risk": 0.07843137254901962, "preDefenseRisk": 0.07843137254901962, "attacks":
[["LEAF2", "Leaf 2"], ["LEAF3", "Leaf 3"]]},
{"risk": 0.06535947712418301, "preDefenseRisk": 0.09150326797385622, "attacks":
[["LEAF2", "Leaf 2"], ["LEAF4", "Leaf 4"]]}]}

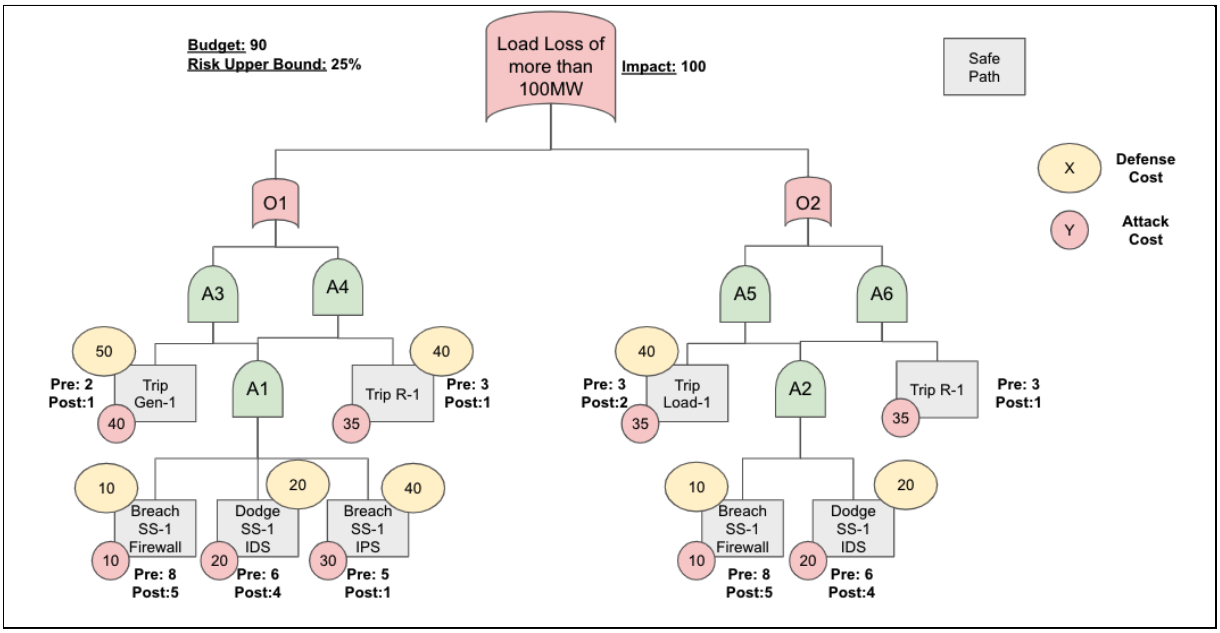
The expected analysis matches the actual output.

3.5 Case Study

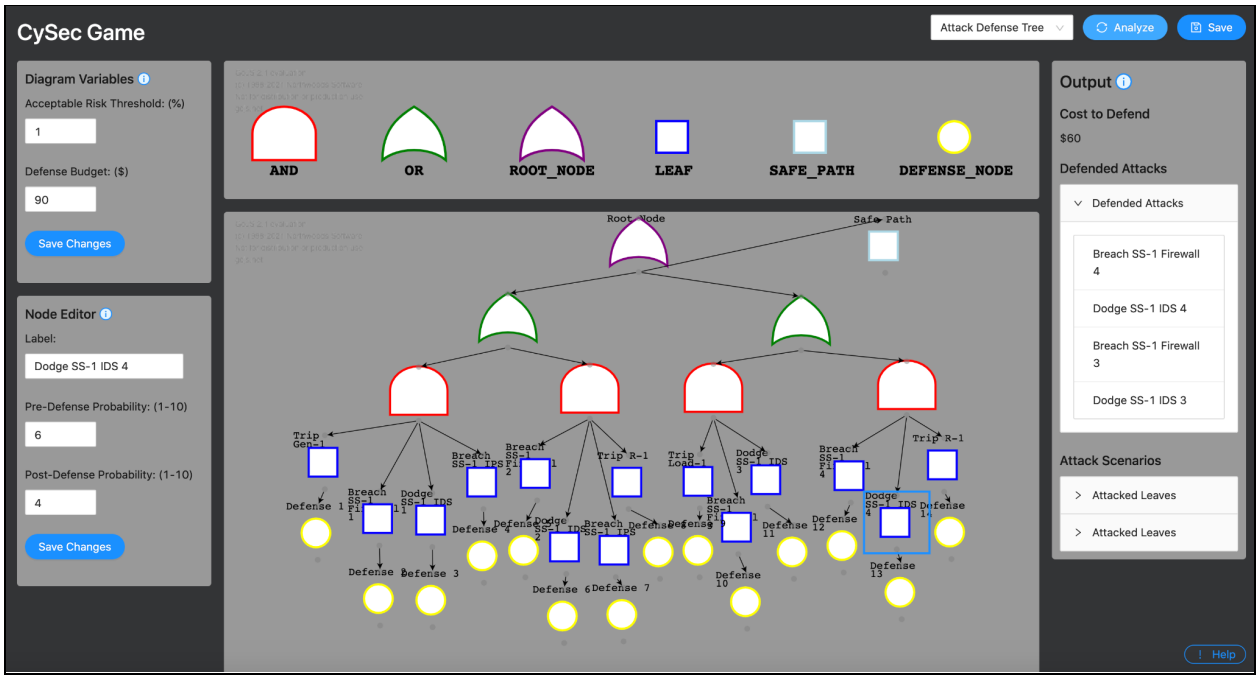
(Case Study provided by Burhan Hyder - Graduate Advisor)



Given is a sample real world example of a power grid. The grid is generating power and there are several touchpoints where we see cyber and physical defenses that a malicious actor may attempt to compromise.



Here we see the attack/defense tree representation of the given power grid system. Each attack node is given a pre and post defense probability in addition to an attack and defense cost. The system is now given a budget and upper risk bound and an impact value to the system if compromised.



Lastly, we can see the same tree represented in the CySec tool. In this case, the attack/defense algorithm was used and found that the optimal defense strategy was to defend the four attacks listed for a cost of $60. There are also drop downs for the attack scenarios that were defended.

4 Related Works

Previous work in cybersecurity modeling has focused on using attack trees and Petri nets to model networks and their associated attacks and countermeasures. Attack trees are the most basic model; implementing an attack tree model is straightforward, but the model fails to capture all the complexities present in real-life cyber-physical systems. An extension of attack trees are attack-defense trees. These models are able to model more properties of a system, but are still limited in modeling dynamic attacks and countermeasures. A master's student working with our client/advisor previously developed a Petri net-based tool that allows users to draw a model diagram of a system and carry out security evaluations. This system more accurately models the dynamic nature of attacks and countermeasures.

There are a number of commercial and open-source products available to perform cyber security modeling. Each product has its own focus and is not suited for every application. For example, the CALDERA framework[1] allows users to run simulated breach-and-execution scenarios. This tool allows users to automate a simulated attack on their system to identify vulnerabilities. CyberX[2] is designed to passively scan an IOT/ICS network to identify vulnerabilities specific to IOT/ICS systems. This is related to the focus of our project, which looks at power systems.

A key difference between our tool and the two commercial frameworks mentioned above is that like the PENET tool, our tool will allow users to model and analyze a network without a direct connection to the network. In mission-critical environments like power systems, this can be a benefit because vulnerabilities can be identified without risk of the assessment tool disrupting the network. Our tool will also implement a number of different analysis engines, including the attack tree model and other, more advanced game theory-based models.

5 Conclusion

Over the past 2 semesters, we have designed and implemented a prototype network security tool that allows users to model threats and learn what investments should be made. Currently, users can:

- Run the tool locally
- Generate a network diagram in our web application
- Process the diagram using an attack tree, attack-defense tree, or game theory engine
- Learn what the suggested investments to the system are

The next steps for the project include:

- Further refinement of the analysis engines, particularly the attack-defense tree engine
- Improving the user experience by adding features like uploading and saving diagrams
- Implementing additional analysis engines

6 References

1. CyberX. (2020). IoT & ICS Security. CyberX. Retrieved 10 18, 2020, from https://cyberx-labs.com/
2. The MITRE Corporation. (2020). CALDERA. The MITRE Corporation. Retrieved 10 18, 2020, from https://www.mitre.org/research/technology-transfer/technology-licensing/caldera

Appendix I - Operation Manual

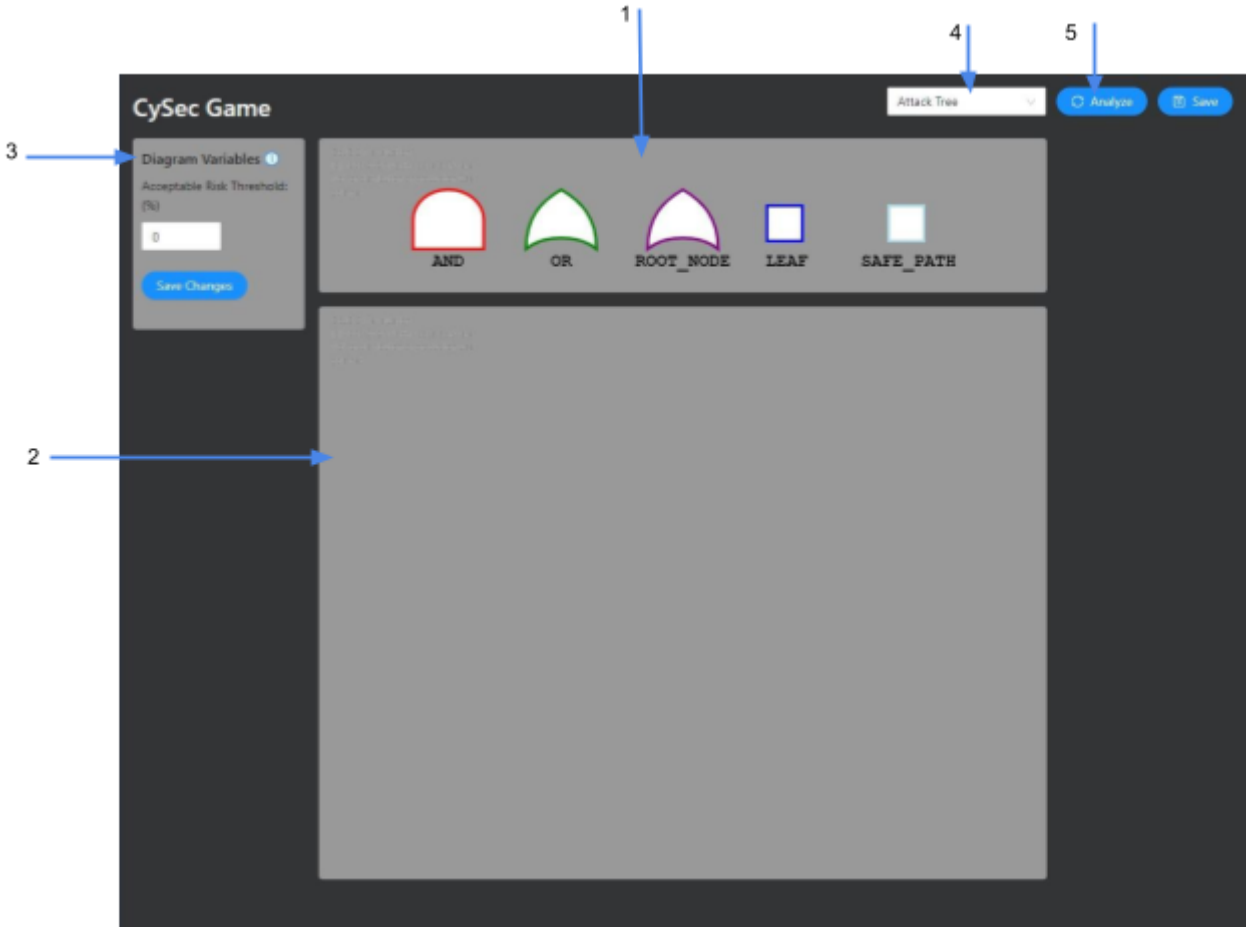
I.1 Setup

Running the application requires the user to have node.js and python3 installed on their system.

To run application:

- Download project from GitLab and open a terminal in the project
- Navigate to the backend folder
  - `cd /backend/api`
- Install dependencies
  - `python3 -m pip install Django djangorestframework django-cors-headers nashpy`
  - `python3 -m pip install nashpy`
- Start backend server
  - `python3 manage.py runserver`
- Navigate back project directory and then to frontend2 folder
  - `cd /frontend2`
- Install dependencies
  - `npm install`
- Start frontend
  - `ng serve --open`

I.2 App Layout



I.2.1 Palette

CySec's Palette displays all the necessary nodes to create a tree. Which nodes are available for you is determined by your selected analysis method. To use a node in your diagram simply click and drag a node to the diagram area. The node won't be visible for a brief moment while in between the palette and diagram, but it will reappear once your mouse enters the diagram area.

I.2.2 Diagram Area

This is your canvas for creating diagrams. Dragging nodes from the above palette renders them on the diagram canvas. Once nodes are placed in the diagram area you're able to scroll around by clicking and dragging within the canvas. If your diagram becomes too large to work with, holding the Control key while tapping the - or + keys on your keyboard will zoom in and out.

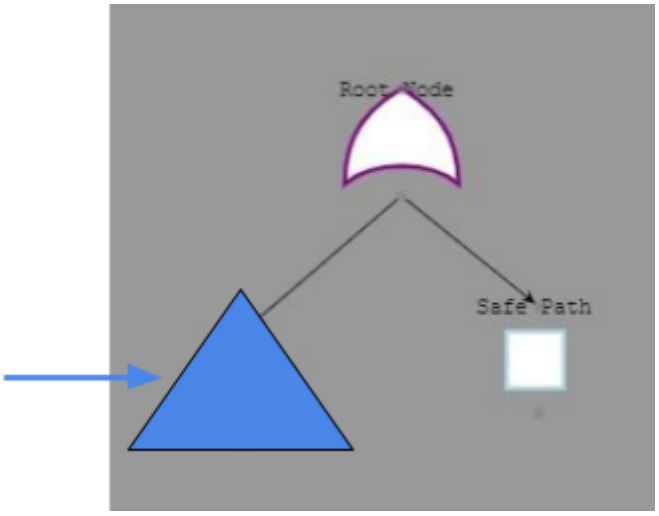
I.2.3 Diagram Variables

This form represents the global variables for the entire diagram; the fields available to users will change depending on their selected analysis method. To use CySec's forms simply click in the input boxes and enter a desired value, however values won't update until you click the 'Save Changes' button.

I.2.4 Analysis Engine Selector
This drop down menu selects the desired server side analysis algorithms to be run on your diagram. This selector also modifies the available nodes within the palette and which input fields are available on forms.

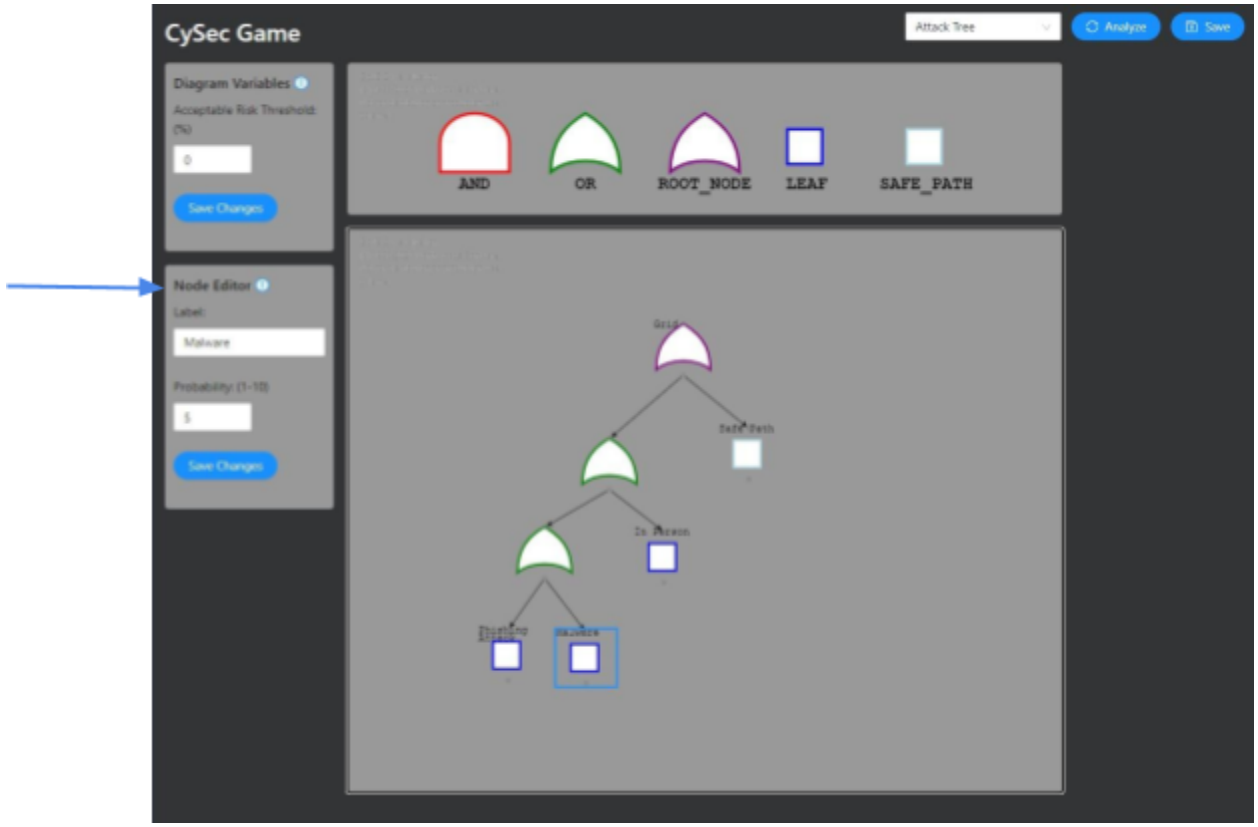
I.2.4 Analyze Button
Once you've completed your diagram and filled out all the diagram and node form fields click this button to submit the data to the server for analysis.

I.3 Creating a Diagram



For the analysis algorithms to give correct output all diagrams must follow the format featured above. A tree's top level node, must be the Root Node component and it must have a Safe Path child node. The Safe Path node represents the likelihood that an attack doesn't occur or that the attacker fails on all attempts. You can begin constructing your system's tree at the same level as the Safe Path; your tree is represented by the blue triangle above.

I.3.1 Attack Tree



After assuring your diagram contains a Root Node and a Safe Path you are free to construct your tree. With Attack Tree selected as the engine your tree will follow a typical tree structure with Leaf Nodes representing your system's weaknesses or potential attack scenarios. By clicking on a node (excluding the And and Or operators) you will see a Node Editor form appear on the left side of your screen. This form represents that specific node's data. The required data will differ for each analysis engine. You are able to edit the Leaf's label and give it a probability that it will be compromised on a scale from 1-10.

I.3.2 Attack Defense Tree

By selecting Attack Defense Tree from the drop down in the top row a Defense Node becomes available within the Palette and new form fields are shown. Now each Leaf node must lead to a Defense Node; these defenses are given a label and a cost associated with their defense. A new Defense Budget field is also required within the Diagram Variables form. The Leaf Nodes now require a Pre-Defense Probability as well as a desired Post-Defense Probability.

I.3.3 Game Theory

A diagram for Game Theory looks no different than one for Attack Defense Tree, the only changes are within the forms. Leaf Nodes now require just one probability and a Cost to attack; alternatively Defense Nodes have a new Cost to defend field.

I.4 Results

After your tree is complete with all inputted values and a valid structure click the Analyze button located in the top-right of the screen. Once the data returns from the server a new Output section will appear to the right of the diagram with your results.

Attack Tree Output

Attack-Defense Tree Output

I.4.1 Attack Tree

Pictured here are the results from an Attack Tree simulation. Based on the diagram you created, the app will show you all potential scenario combinations that will result in system compromisation. Additionally you will see the risk that this scenario holds and the probability that it will occur.

I.4.2 Attack Defense Tree

When using the Attack Defense Tree engine you will see a new Defended Attacks section that shows attacks the system was able to defend with your available budget. The Attack Scenarios also now show the calculated risk after defenses and before defenses.

I.4.3 Game Theory

Using the Game Theory engine will show you a new Recommended Investments panel, this panel shows which defense nodes you can invest in and how much to invest to receive optimal results. Additionally you are shown Payoff Matrices and Nash Equilibria for your diagram. These mathematical models are how the engine is able to recommend defense investments.

Appendix II - Other Considerations

This project is a working prototype, however all functionality is not complete. It is the goal of the client that this project be handed off to either another senior design team or a research group in the coming semesters. Some features that still need to be implemented are:

- Coloring attack scenarios based on the acceptable risk threshold.
- Highlighting relevant attack nodes on the tree when the user clicks an attack scenarios in the output column.
- Await and async functions should be used in communication with the backend to smooth out the user experience.
- Possible integration with software to build a tree and input data from another system.
- Possibility of adding login to store the user's attack tree diagrams.

As the CySec tool is currently used locally through a web-browser and does not currently require a database we do not have to deal with many common security concerns. The data inputted by the user is sent to the backend where it is not logged or stored in any way and then analysis is sent back to the frontend for use by the same user. Since the tool is used locally, we also do not have the risks associated with sending data over a network. However, in the future the app may store attack diagrams for the user and thus security measures will need to be implemented to ensure their defense structures remain confidential. Similarly, if this application is deployed over a network, steps will need to be taken to ensure the privacy and integrity of the data in transit.