Project Update 2: Progress in Prioritizing Patch Management Using Quantum-Inspired Vertex Cover Algorithm

Alli Ajagbe, Ayush Sheth

Project Methodology

This project leverages the Minimum Vertex Cover problem and the quantum computing framework Classiq to analyse vulnerability data represented as a bipartite graph. A dual graph capturing connections between vulnerabilities is constructed. By solving the Minimum Vertex Cover problem on the dual graph using the QAOA algorithm, we aim to identify a minimal set of vulnerabilities to patch, effectively disrupting potential kill chains.

Latest Updates:

More Literature Reviewed

As mentioned earlier, the both of us are completely new to cybersecurity and had to take a good amount of time to understand exactly how patching management is performed. For this, we understood how kill chains function, what are connectivity dual graphs, and how we can use vertex cover graphs to eliminate these kill chains. For this purpose, we learnt and understood what the minimum vertex cover problem is, and further what classical annealing optimization algorithms are, which are used to solve the minimum vertex cover problem. Upon extrapolating, it leads us to quantum annealing optimization and hence QAOA – Quantum Approximate Optimization Algorithms. All of this has been meticulously reviewed from over 10 websites and papers, and has been updated on the Excel sheet we are utilising for review documentation. The definitions and explanations for the same have been updated on the 'description.md' file on out GitHub.

Building Optimization Model using Graphs

We have utilised the *pyomo* package in python for its inbuilt functionalities to create a model for the graph system and kill chain as described in the Examples section on our git. As we want to utilise Classiq's online IDE for the project, it is important to note that the *pyomo* object created will directly fit into the model we make for the system on Classiq.

This code for the *pyomo* object creation can also be located on our git in the classiq.py file. Following along the code, it is visible that we have defined an object '*mvc*' as the pyomo object we want to utilise.
As a *pyomo* object, the model contains – a binary variable declaration in *model.x*, to check whether the variable, which is a node, is chosen in the set; a constraint rule to ensure all edges between the selected nodes being covered; and an optimization objective rule to minimize the

number of nodes selected.
Following from this, the optimization of selected nodes are the ones that require patching, and it is the minimum number of nodes in the network that hence require to be patched to avoid all kill chains.

Making Quantum Model to Solve

Further, to solve the model using QAOA (Quantum Approximate Optimization Algorithm) as we have proposed, we need to create a quantum circuit model for the *pyomo* object. This was done by utilising the *classiq* package available for python.

The quantum model we built therefore, is actually a Classiq model, that can be easily interpreted and worked upon on the Classiq IDE online.

Following the same code further in the classiq.py file, we incorporate classiq's QAOA algorithm into the *pyomo* object.
First step is to generate a quantum and classical config for the implementation of the QAOA algorithm using the Classiq combinatorial optimization engine; the *qaoa_config* and *optimizer_config* objects are used for that purpose.
Second, we create a quantum model using the *pyomo* object, and store it in the *qmod* variable, as can be followed from the code.
Post this, thirdly, we execute the model and the algorithm, and the classiq package helps us write the final *qmod* object out into a file readable by the online IDE for further analysis.

Further Plans for the Project:

1. As we can now generate solutions using QAOA, we will refine them by potentially considering additional factors or constraints not directly captured in the initial model.
2. We further want to plot out the solutions as graph, and demonstrate how patching works and saves the infrastructure from kill chains (attacks).
3. We also want to extrapolate and present the minimum vertex cover problem to demonstrate how it translates to vulnerable infrastructure, in a way that it can be understood by laypeople.
4. We are trying to work through and understand the Classiq IDE better, to help us with analysis of the solutions, as it already has integrated multifunctional menus for analysis of quantum models and circuits.
5. We want to try the process out for larger models, which there is a high possibility of the systems not being to handle.
6. Again, we want to meticulously document all of this on the git, so that it is readable and replicable as a project later on, and also something that can be worked upon for future aspects beyond the scope of this project.

We would once again like to point out that we are still learning whatever we will have to implement for the project, so kindly point out if there are any discrepancies, and we will take due note to correct them.

Do reach out to us in case of any questions!