Graph Neural Networks for Polymorphic Virus Detection

Nathan Waltz, Kaitlin White, James Halvorsen and Assefaw Gebremedhin
School of Electrical Engineering and Computer Science, Washington State University

Introduction

- Polymorphic viruses are self-modifying viruses that slightly modify their source code each time they self-propagate.
- While polymorphic viruses modify their source code, they still perform the same task.
- The fact that the source code is modified makes these viruses exceptionally difficult to detect.
- Using reverse engineering tools such as Ghidra and Cutter, we can generate a global call-graph of a program. We can then use graph neural networks to determine graph similarity and develop a signature for the program and determine whether a program is infected with a polymorphic virus.

Reverse Engineering

- There are methods to reverse engineer polymorphic viruses through platforms such as Cutter.
- Cutter is a tool that reads the binary code of an executable and re-creates it in assembly so it can be analyzed at a higher level that is easier to understand.
- Cutter has a tool to create global call-graphs which show the calling relationships between a program's subroutines.
- Then using graph neural networks, the global call-graphs can be loaded into memory to check for similarity between two programs.

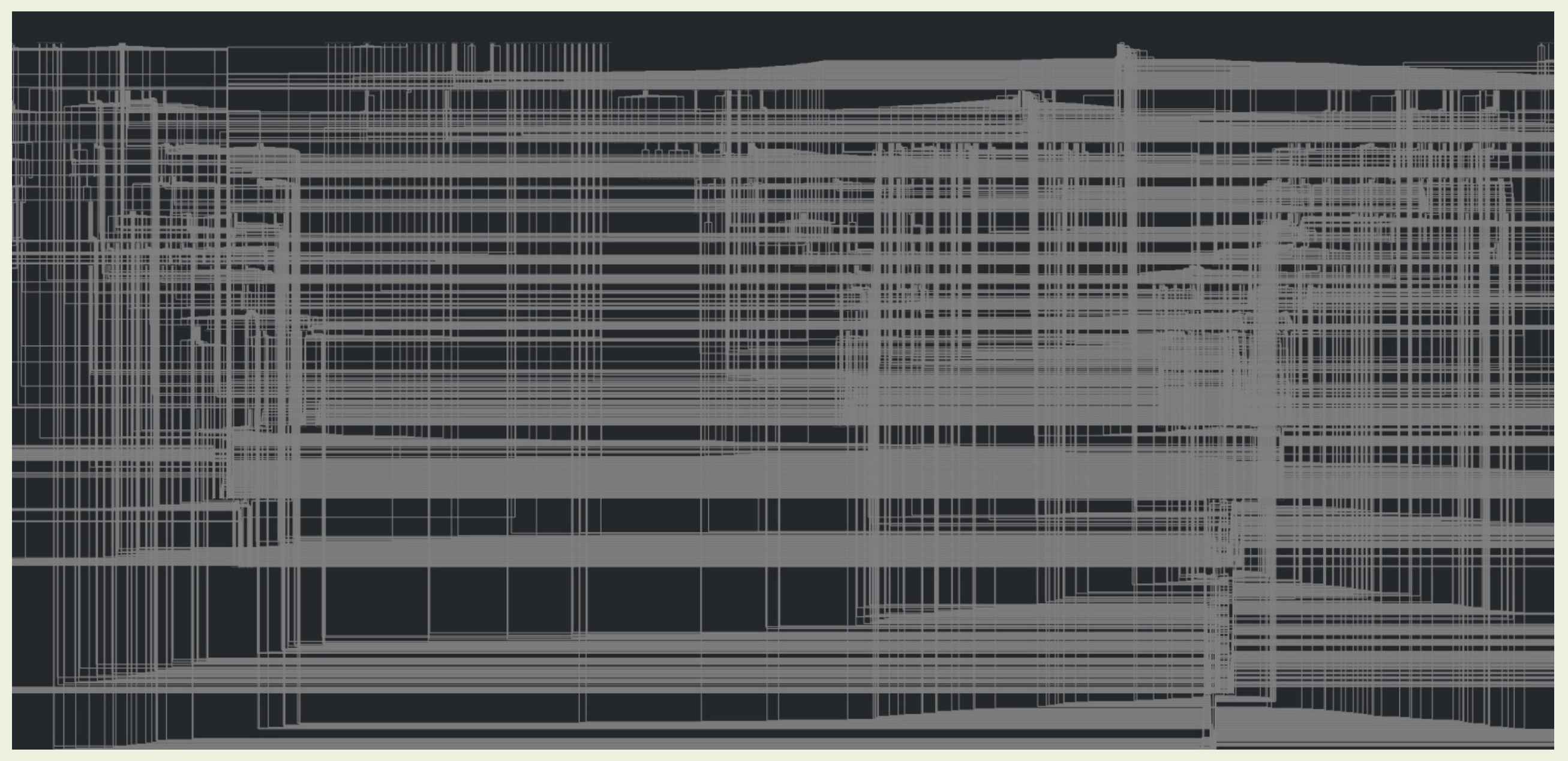

Figure 1: Sample control flow graph

Graph Neural Networks

- Graph similarity has been one of the main applications of graph neural networks
- For example, SIM-GNN is one research project that investigated finding program similarity using graph neural networks.

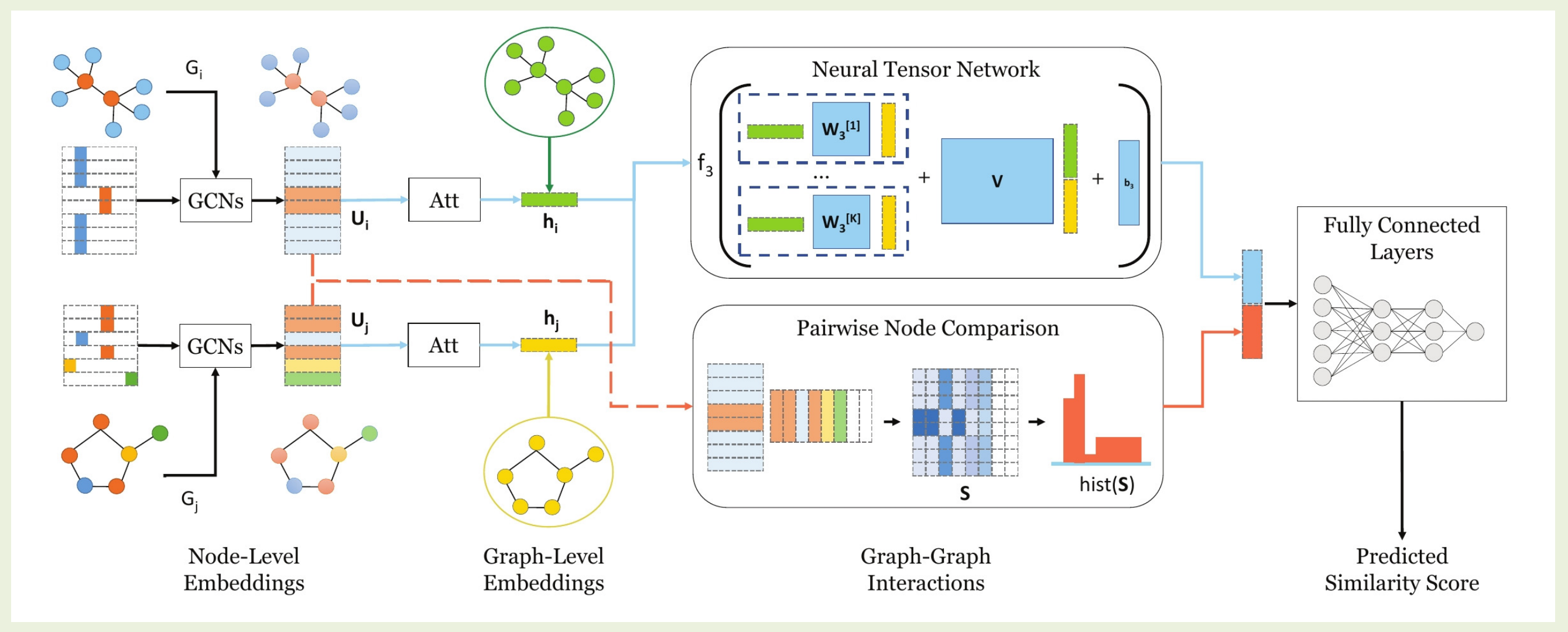

Figure 2: SIM-GNN graph similarity detection diagram

- Using SIM-GNN or a similar program, we can determine whether two programs are the same – and expand this to the realm of polymorphic viruses to determine whether two infected programs are infected.

Experimental Design

The Data

- The datasets will use a mixture of benign and non-benign global call graphs of different executables.
- The graph neural networks used for training were implemented using Python and modeled off SIM-GNN.

Training Phase

- The datasets will be labeled, and this will allow us to apply supervised machine learning algorithms to the data.
- The data will be split into three distinct sets – the training sets, the validation sets, and the testing set.
- The GNN will hopefully be able to determine if a program is infected with a polymorphic virus.

Testing Phase

- The machine learning models will be tested on unseen data to determine whether overfitting has occurred, and how well the model generalizes.

Research

- While this project is still ongoing, there has been quite a bit of research on polymorphic virus data.
- Anomaly detection using matrix profiling was investigated first, but this proved to be an ineffective method given the shape of the datasets.
- The binary executable files of different programs were reverse engineered and stored into global call-graphs for usage with graph neural networks.
- This will allow us to incorporate more established cybersecurity techniques such as reverse engineering with more cutting-edge data science techniques such as graph neural networks.

Background on Polymorphic Viruses

- Polymorphic viruses are types of malware that can change programs or files on a computer without being noticed.
- These viruses can also repeatedly make changes, disrupting the ability for virus-protection software to stop them, if it can even be found in the first place.
- This can be harmful to computers because once an infected program is executed, it becomes incredibly difficult to repair once the virus replicates itself.
- The damage that can be done spans from loss of sensitive data or files to complete failure of a computer's operating systems and programs entirely.

Deep Learning Architecture

- The datasets will be analyzed with a SIM-GNN architecture – which is a multi-layer graph neural network that utilizes batch MSE loss to derive gradients and converge to a local minima.
- Programs will be mapped into global call-graphs, and then these global call-graphs will be analyzed by the SIM-GNN model.
- For each program, the model will compute similarity between a program and sample programs infected with polymorphic viruses. We will use this similarity to determine whether a program is infected with a polymorphic virus.

Potential Impacts

- Polymorphic viruses are dangerous to computers because they can be extremely hard to detect.
- These types of viruses rely on altering programs in small ways to be undetectable to standard virus-protection software available on most computers.
- Training the graph neural networks to effectively detect polymorphic viruses would have major positive impacts in computer security.
- Networks that can detect these essentially undetectable viruses before program execution would prevent potential harm or loss to a computer or sensitive data before it can even occur.
- As this research continues, it could be used to protect specific types of programs/data or become standardized in all virus protection software available to any user.
- Effective detection would also further secure high-risk organizations from criminal activity.

Future Work

- This work could be expanded to more complex programs and/or different virus types to provide a wider range of virus detection.
- The results of comparison between the benign and non-benign graph types could be narrowed to look for specific types of discrepancies.
- The trained graph neural networks could be extended to more generalized cases.

Acknowledgements

- The author(s) are grateful for funding from the Griffiss Institute under contract No. SA10012021MM0336.
- This work was supported by the CYSER program and WSU's Computer Security Group.