Section II: Methodology

Role of Student vs. Mentor

Although I have sought advice from some professors, most of the work has been done by me. I have been working on this project, including the pre-research, since September, totaling more than 115 hours of research, documentation, and experimentation.

Equipment and Materials

Most programming was done through the IntelliJ IDE, on a Lenovo Yoga laptop with a 10th generation Intel Core i7 processor and 16 GB of RAM. Java was used for data generation and processing, while Python and the PyTorch library were used for machine learning. Google Colab was used to train the model due to its GPU capabilities.

Techniques

All code can be found here: https://github.com/gouda64/neural-chain-preimage-attack.

Data Generation

The SHA-1 hash was implemented in Java. Binary strings were randomly generated and inputted into the hash, and the internal state history was recorded in CSV format. The length of these strings was 432, to ensure that the data is only one block while accounting for padding, as SHA-1 operates in blocks of 512 bits (Dang, 2015). To circumvent memory restrictions and make use of a greater number of training samples for individual layers, only the relevant two layers were recorded in some cases. Data with a restricted, nonrandom input space (hereby referred to as keyspace) was also considered, as it potentially improves accuracy by effectively making more information available to the neural networks, as shown by the excellent results in So (2020). To create a restricted keyspace, alphanumeric ASCII strings were randomly generated, then converted to binary: as alphanumeric characters do not represent all possible ASCII characters, the resulting binary strings had a skewed bit distribution. These strings were then inputted into the hash and serialized. In total, a dataset of 100k full histories with unrestricted keyspace, a dataset of 100k full histories with restricted keyspace, and a dataset of 1 million layers 2 and 3 with unrestricted keyspace were generated.

Neural Network Creation and Training

The PyTorch library was used to parse data and create the algorithms necessary to train and test the machine learning models. It was also used to create several models: a multilayer feed-forward neural network, a recurrent neural network based off of Greydanus (2017), and a feed-forward network on fuzzy data based on Goncharov (2019). The code and dataset were then uploaded to Google Colab for training. Specific hyperparameters, such as batch size, learning algorithm, layer depth, learning rate, and dataset were experimented with while testing.

Linking and Postprocessing

The models that reached sufficient accuracy would have been saved and linked in a data pipeline using Python. Inputs would have been randomly generated, hashed, and entered into the chain to reconstruct the internal state history of the hash, from which a partial preimage could then be extracted and compared to the known preimage. Unfortunately, this stage was never reached due to none of the models reaching sufficient accuracy.

Analysis of Data

No statistical tests were undergone to analyze the data produced. Since this project focuses on theoretical combinations of techniques, implementational neural network results are not being compared and do not need statistical analysis—rather, they serve as a method of evaluation for the techniques themselves. Accuracy and loss were the primary benchmarks used.