CS G513 Network Security Project

Second Semester 2022-2023

Important Instructions

1. The information and research that you will be referring to during this project should be publicly available and properly cited in your report.
2. The collected information should be used for ethical purpose of the study.
3. Any attempt to (i) gather information which is not publicly published, or (ii) using the collected information/ developed methods for unfair purpose, will be considered unethical and strictly dealt as per the guidelines for the punishment of unfair means.
4. Initial selection of the project topic and group details should be provided at the below link on or before **21 March 2023, 10 PM**.
   Link to submit project topic and group details: https://forms.gle/2gBqR5xtHsaSzzt48
5. Final project submission deadline is **25 April 2023, 6 PM**. This should include all working files zipped in a folder with a readme. The folder should be renamed to project number (from the below list), followed by the first name of all group members, e.g. <P1>_<name1>_<name2>_<name3>_<name4>.
6. A separate project report not more than ten pages providing one paragraph description of the selected project and a detailed description of the developed methodology and each group member's contribution is also required to be submitted. You can take the help of step-wise description/ figures/ flow-charts here.
7. Final project submission link: https://forms.gle/o8HaFCb5TfA6TKMy8
8. More details regarding demonstration will be provided in due course of time.

The objective of this project is to get connected to a real-world cyber-security problem and work on its probable solution. To accomplish this task, you are required to select one of the project titles provided below and implement it as a working model. The schedule to demonstrate results under certain use-cases will be informed in due course of time. Group formation is permitted given that the maximum members in a group should not exceed four. The role of each group member should be well justified in terms of the contribution to the project.
Note: Any one student from each group is required to fill the Google form on or before 21 March 2023.

P1. Design of light-weight encryption algorithms

Description: Privacy awareness regulations and emergence of data silos have created new challenges to the traditional way of data encryption. Traditional encryption schemes consider data in small chunks to process. This approach becomes computationally expensive with standard security key sizes (1024-bit or more) when the underlying data size is very huge. In this project, you are required to address this problem by the development of light-weight encryption algorithms that are efficient both computationally and storage wise. The project should consider standard encryption algorithms (at least two) with varying keys sizes (e.g. 1024-bit, 2048-bit, and 4096-bit) and develop their efficient versions for plaintext data of at least 1 GB. The computational time as-well-as storage capacity of the traditional schemes and developed variations should be well demonstrated with a notable difference. Here, it is important to note that the security of the selected encryption schemes should not be compromised during the development of its light-weight counterpart. This part should be well addressed by providing mathematical proofs in the report and a working demonstration. Even if there is a minor difference in security standards of the developed schemes and their actual ones, it should be well documented with mathematical proofs and a working demonstration. Different standards including IND-CPA, CCA, and CCA2 should be implemented to demonstrate the security of the developed schemes under various use-cases. Certain use-cases will be provided at the time of demonstration to test this functionality.

P2. Data-driven adversarial user detection in a distributed e-healthcare environment

Description: Consider a distributed e-healthcare system for patient health prediction among *n* users. Initially a common baseline ML model is shared with all users in the system. Here, each user is a participating entity with its own private data that is used to train the received ML model locally. Once the training is done at each user, the trained models (total *n* models, one model from each user) are required to be averaged at a central repository. This makes locally trained models more powerful by integrating them to a single global model. In this scenario, if any user is compromised by an adversary or turns out to be an adversary, it would badly affect the performance of the global model. The probable outcomes would be wrong prediction for patients test samples (targeted misclassification), decreased accuracy of the global model, etc. **You are required to find such adversarial user(s) at data level (data which is locally available at each user) and develop a working solution for the same.** Here, the adversary should be identified and dropped early before model averaging so that a sufficient difference on global model performance before and after including the adversarial client should be visible. A probable solution may be finding the difference between received model updates and tracing back to the suspicious user once its locally trained model is found to be differently than others. At user level (suspicious user/ top two suspicious users), different data auditing parameters/ metrics can be worked out to get into deeper levels and further rectify its **data** for adversarial behavior. The methodology workflow is described below.

1. Initially, the dataset should be randomly distributed to *n* number of users. This will help to simulate a distributed system among *n* users with their own data.
2. Once the users are populated with their data (available locally now), share the model with all of them. After receiving the model, each user should train it using its own local data.
3. After model training at each user (with its own local data), the user should revert back the trained model to the central authority (where all other models are also collected).
4. When all trained models are received at the central authority, the developed algorithm should work to identify adversarial user at data level.
5. Finally, model averaging should be done at the central authority by dropping the adversarial user's trained model and combining remaining models. This would include computing the average of the trained models.
6. For simulation purpose, the adversarial user, *A*, can be explicitly created by poisoning its local data through label flipping, adding random samples with noisy data, etc. Two cases are described below that should be considered for this purpose:
    a. Case I: Modifying forty percent of the local data available at *A*.
    b. Case II: Modifying ten percent of the local data available at *A*.
7. The results of the global model before and after merging adversarial user should be demonstrated.

The dataset and model references required to accomplish this project are provided below.

Dataset Link: https://www.kaggle.com/datasets/rischan/diabetes-dataset

The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. The classification task is to predict whether a particular patient will readmit within 30 days. This is a binary classification problem with labels as `<30' and `>30' for re-admittance. The dataset contains 45,715 entries, with 11,066 entries for the positive class (<30 days) and 34,649 entries for the negative class (>30 days).

Model Description: You can use a simple CNN architecture (or customize it as per your convenience).

References for model designing:

1. https://www.kaggle.com/code/vishwasgpai/guide-for-creating-cnn-model-using-csv-file/notebook
2. https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial

P3. Model-driven adversarial user detection in a distributed e-healthcare environment

Description: Consider a distributed e-healthcare system for patient health prediction among *n* users. Initially a common baseline ML model is shared with all users in the system. Here, each user is a participating entity with its own private data that is used to train the received ML model locally. Once the training is done at each user, the trained models (total *n* models, one model from each user) are required to be averaged at a central repository. This makes locally trained models more powerful by integrating them to a single global model. In this scenario, if any user is compromised by an adversary or turns out to be an adversary, it would badly affect the performance of the global model. The probable outcomes would be wrong prediction for patients test samples (targeted misclassification), decreased accuracy of the global model, etc**. You are required to find such adversarial user(s) at model-level (at central server after averaging) and develop a working solution for the same.** Here, no data-level auditing is required and the adversary should be identified at the model level only. A probable solution may include tracing back to the suspicious user once its locally trained model performs differently than others at central authority. For this purpose, different model evaluation parameters/ metrics or a weighted combination of more than one metrics can be worked out to get into deeper levels and further rectifying the suspicious **model** for adversarial behavior. The methodology workflow is described below.

1. Initially, the dataset should be randomly distributed to *n* number of users. This will help to simulate a distributed system among *n* users with their own data.
2. Once the users are populated with their data (available locally now), share the model with all of them. After receiving the model, each user should train it using its own local data.
3. After model training at each user (with its own local data), the user should revert back the trained model to the central authority (where all other models are also collected).
4. When all trained models are received at the central authority, model averaging should be done. This would include computing the average of the trained models.
5. Now, the difference between baseline global model (initially shared model) and merged global model should be computed. Also, one-to-one difference between baseline and each locally trained model should also be computed. Here, different model evaluation parameters/ metrics or a weighted combination of more than one metrics can be worked out to find the most devastating locally trained model as the suspicious user.
6. For simulation purpose, the adversarial user, *A*, can be explicitly created by poisoning its local data through label flipping, adding random samples with noisy data, etc. Two cases are described below that should be considered for this purpose:
    a. Case I: Modifying forty percent of the local data available at *A*.
    b. Case II: Modifying ten percent of the local data available at *A*.
7. The results of the global model before and after merging adversarial user should be demonstrated.

To accomplish this task, the same dataset provided in the previous topic (P2) should be used.

P4. Code-breaking using intelligent differential attacks

Description: Differential attacks analyze variations in the input plaintext with variations in the output ciphertext to find the desired key or plaintext message. In this project, you will be required to develop a GUI based interactive model that can break a given ciphertext using differential attacks. The model should take *n* number of ciphertexts (*n* should not be greater than 10) and the underlying encryption algorithm, perform differential analysis, and output the plaintext(s) and decryption key. For implementation purpose, Playfair cipher, Vignere cipher, DES, and RSA algorithms should be supported by the model. A minimum key size of 128-bits for Vignere cipher and DES, and 1024-bits for RSA algorithm should be considered.

The methodology workflow is described below.

1. The GUI should have two different modules: First module for ciphertext generation and second module for differential analysis.
2. Initially, first module should be referred by selecting the encryption algorithm and generating its respective key(s). The generated keys should be displayed as well.
3. After key generation, the required number of ciphertexts should be generated by giving respective plaintexts and secret key(s) as the input. The generated ciphertexts should be stored in a separate file.
4. Now, the second module should be referred to take input from the stored ciphertexts from the file and the corresponding encryption algorithm should be selected.
5. The outcome of the second module should be the secret key for decryption and the underlying plaintext(s).
6. The outcome of the intermediate differential attack methodology should be visible in a separate panel. This can include graphs and bar charts to demonstrate ciphertext to plaintext analysis visually.
7. Attacks under the relevant standard(s) IND-CPA, CCA, CCA2, etc. should be implemented to demonstrate security vulnerabilities of the developed schemes under various use-cases.
8. Certain use-cases will be provided at the time of demonstration to test this functionality.