INSE 6130 /2

Operating System Security

Re-implementation of "Effective Attacks and Provable Defenses for Website Fingerprinting"

Progress Report

Akshaya Mahadevan	**7244207**
Lina Nouh	**9504540**
Manisha Mavullapally	**7607024**
Nishanthini Suresh Babu	**7398271**

Table of Contents

Abstract

In today's world, the web has become the main medium in which information is transferred. A lot of users access different websites in order to gain access to different information. These users who browse the web do not expect their privacy to be breached. Users who are aware of this possibility of breaching their privacy may use low-latency anonymity networks in order to protect their privacy from different types of attacks such as fingerprinting attacks. One such network is Tor. Tor helps in protecting the user's privacy by hiding the information contained in each packet. The user's information can be protected for an anonymous surfing of the internet and it helps to protect the user's location and their activities from government agencies or anyone else. Tor helps in protecting the personal privacy of the users and to have a confidential communication and their internet activities from being monitored by others.

A local, passive adversary can perform website fingerprinting attacks by sniffing for packets being transferred through a network. Fingerprinting is the process of creating the profile of use action of a person in the web. This is basically used for identifying the frequent user of that particular website. This can be done by acquiring the sequence by which the packets are sent. After the acquisition of such packet sequence information, the adversary can identify a certain user's web activity by breaching the user's privacy.

The unit of data that is transferred between the origin and destination is known as packet. Each packet contains information like the packet size, order, and timing. The adversary can use such information to distinctively detect which webpage the user is visiting. Machine learning and classification techniques are used to properly utilize the sequence packet information in the process of identifying the webpage. The fingerprint contains the user's private data such as the user action and the decision about visiting the web pages. The private data may be the user's login information such as username and the password credentials. These private data are stored in the browser's history and can be breached and thereby resulting in a severe privacy threat.

In order to prevent such privacy breaches, it is mandatory to store the browser history containing the user's private data in an encrypted form so that others cannot access it. When the user's data is in an encrypted form, it will be difficult for an attacker to read the data without knowing the suitable decryption methods and hence would prevent the breach of the user's private data stored in here. In this project, we implement some defenses that act as protection against website fingerprinting attacks. Then, we attempt to implement the fingerprinting attack even after the defenses are applied.

Planning and Preparation

Before we get started with the project, we decided to plan our course of action. We planned to begin by reading the paper "Effective Attacks and Provable Defenses for Website Fingerprinting" thoroughly in order to properly understand its contents and methodologies of the paper. Each team member read the paper individually. Then, we had several meetings to discuss the ideas being presented in the paper so that we could have a better understanding of the paper. This helped us to efficiently implement the project. We also did some research to improve our understanding of some concepts that we were not familiar with. One such research was about the Tor browser. Tor is an anonymity network that allows people in it to improve their privacy standards and help them prevent security breaches of their private data. It is used by many people from generic browsing to a very confidential issues.

Our next step was to download Tor and have first-hand experience using it since it is used as a defense tool against website fingerprinting attacks. After we got a good understanding of the paper and what we need to implement, it was time to make some technical choices. We decided to use gmail for communication between the group members, and google drive for sharing the documents and collaborating on writing the artifacts.

We decided to implement this project in Java since it will be more efficient and comfortable as all group members have previous experience in Java programming. Then, it was time to set up the work environment. After that, we elicited the requirements which we have to implement in order to complete the project. We decided to start by implementing the decoy pages defense and the kNN attack.

Reaching Out to the Authors

For the better understanding of the project implementation, we contacted the authors of the paper. They were kind enough to provide the information about the project methodologies and they helped us by providing the data sets needed for the reimplementation of the project. The dataset is a huge collection of packet information that had been collected from over 100 websites and over 9000 open world instances. Collecting this amount of data is impossible in the limited period of time that is given to us.

The information provided by the authors and their instant help, made us understand the techniques and effectively reproduce them in the project. This assisted in greatly improving the overall efficiency of completing the project. By understanding the concepts and methodologies that was pointed to us by the authors, we were able to use our ideas and effectively implement them for the betterment of the project.

Setting Up The Environment

The first step we had to do before starting to code was to set up the work environment. We needed to use a medium through which we can share our thoughts and ideas. Keeping this in mind, we decided to use GitHub in order to collaborate in working on the program code files. GitHub is a Repository which allows many software engineers to work on a single project without the need to share a common network. It manages the changes made to the large collection of document and updates them in a regular manner.

Using GitHub, we created new individual accounts for the members of the team. We also created a new git repository to share the work which would be contributed by the team members on this project. Each group member cloned the repository locally on her computer in order to work on the project then commit any new contributions. We installed the software required for developing Java code. The group members are working on Eclipse to develop the code. Eclipse is an IDE (Integrated Development Environment) which has a base workspace and plugins which is used for developing applications. Eclipse can be used to develop applications in many programming languages. For our Project we use Java as the programming language.

Architecture and Design

In order to build the system, we need three main components: the packet sequence data, the defence against fingerprinting, and the fingerprinting attack. The packet sequence data consists of the sequence of the packets that contains the data of the users. The data stored in the packets are usually the size of the packet, sender's IP address, receiver's IP address, data that has to be delivered. These data stored in the packets are used by the attackers for hacking the user's information and thereby creating a security breach of the user's privacy. The defence against the fingerprinting deals with how the attacks can be prevented and how efficiently the packets can be safeguarded from the attackers. The fingerprinting attack is about creating the user profile that consists of the user actions and their personal information such as login information, banking information. These data when collected can lead to a serious threat of fraudulent.

We decided to reuse the packet sequence data from the original implementation of the paper "Effective Attacks and Provable Defenses for Website Fingerprinting". The reason is that it would take too much time and effort to gather this amount of data. The amount of data that was used is taken from 100 websites, 90 instances each, plus 9000 open world instances. Collecting this kind of data in this amount is beyond the scope of this project. We also wanted to focus our time and energy in implementing the fingerprinting defense and attack algorithms since we view them to be the main purpose of the paper and its experiments.

When it comes to the defense against fingerprinting attacks, there are several choices that can be applied. The paper "Effective Attacks and Provable Defenses for Website Fingerprinting" also mentions several possibilities. The other possibilities for the defense are traffic morphing, HTTPOS split, BuFLO, tamaraw, supersequences. We decided to implement the defense of decoy pages. In this defense strategy, a decoy page is loaded at the same time that the intended page is being loaded in order to hide the information and details of the intended page's packet sequence containing the user's private data. This is intended to make it more difficult for an attacker to identify the webpage which the user is visiting and also the personal information the user is providing over the internet in the webpage.

Implementation

The Implementation of this project consists of three main parts which are: the extraction of the feature set, the defense, and the attack. Since we are using Java to implement the project, we are taking advantage of object oriented programming. We created a class for each of these main parts. We created the FeatureExtractor.java class as the class responsible for extracting the set of features for each packet. We created the DecoyPageDefense.java class as the class responsible for applying the decoy page defense to the packet sequence data.

We started the implementation of the Learner.java class which will be the class responsible for applying the kNN fingerprinting attack. Moreover, we decided to implement a Packet.java class in order to represent a packet and hold its details as attributes. We found it to be easier to store, access, and manipulate each packet's information within a packet object. We also implemented a PacketComparator.java class in order to help us in sorting packets which are held in a collection (array or list).

Defense Against Fingerprinting

In Order to protect the user from the fingerprinting attack, the defenses are applied to the client's connection. Defenses are of two types - simulatable and non simulatable. A simulatable packet is that which reads a packet sequence and outputs another packet sequence. This type of defense function does not look at the whole packet contents. Instead, it considers only a part of the packet sequence information such as the packet size, timing, and order. The simulatable defenses are cost effective while the non simulatable defenses have a high implementation cost as they require the access to the client's data and should be implemented on a browser.

On the other hand, the simulatable defenses need the data that a normal attacker would already have. The defenses can also be divided into random defenses and deterministic defenses. If a function returns the same packet sequence as output always when called with the same input packet sequence, then it is known as being a deterministic defense. Random defenses, also known as noise, can be partially removed if an attacker links together different page loads. The defense that we have implemented till now is the decoy pages defense. The decoy pages defense is carried out by hiding the client's information by producing a decoy page and loading it at the same time as the intended page is loaded. The decoy pages defense which we are implementing is considered to be a simulatable and deterministic defense against website fingerprinting.

DecoyPageDefense.java

The decoy pages defense algorithm was used to implement the class DecoyPageDefense.java. In this class, the method createDecoy() is responsible for applying the steps of the algorithm. For the Decoy Pages defense, we use the data set acquired from the authors of the paper and store it in a folder called Batch which we access in the implementation of this class. The Batch folder contains many text files. Each text file represents an instance of one of the websites in the data set. The Batch folder acts as an input folder and the output is saved in a folder that this class creates with the name batchusenix-pdef. Initially, we check if these two folders exist. If the Output folder does not exist, then we create it. If the input folder does not exist, then we print out an error message. As without the input, the process cannot be completed.

Once the Input and Output folders are created and verified, the process starts. Initially, we open each and every input file in the Batch folder in the read mode. On opening the file, the file is read and its lines are stored in an array list. The file is now processed and is stored into another variable. The processing of the file is done for the time and packet direction values stored in the file. The time represents the times when the packet was sent. The direction represents whether the packet is incoming or outgoing. We create packet objects from the Packet.java class and initialize them with the proper time and direction information. Now, we initialize an array list called packets to store the packet objects with the information received and processed. The same process is then repeated with another file acting as the decoy page. Again, packet objects of the Packet.java class are created and added to the packets array list. After that, the processed packets are appended and sorted. The processed data is then written to the output file and the file is closed.

Packets Feature Set

The fingerprinting attack depends on the feature set extracted from information contained in every packet sequence. The feature set is built from the data files we are reusing from the previous fingerprinting attack and defense work which was implemented by the authors of the paper. The feature set contains six important features. From these features we have used general features: the total number of packets transmitted, unique packet length, packet ordering, and bursts of outgoing packets where there are no adjacent incoming packets.

The general features provide information about the total time needed for transmission, and the number of incoming and outgoing packets. The unique packet length describes about the packet lengths of the dataset which can help identify a certain webpage. The packet ordering features provide information about the total number of packets in the sequence and also where the outgoing packets are concentrated. This information is also helpful when trying to identify the webpage each packet belongs to. The bursts information for outgoing packets occurs when there is a sequence of outgoing packets with no two neighbouring incoming packets. This information can be helpful as well in determining the classification of a webpage.

FeatureExtractor.java

In the FeatureExtractor.java class, the steps to extract the features from the packets are implemented. The extract method in this class is the main method that is responsible for extracting the features from the packets. In addition to the extract method, we also implemented some helper methods. The main responsibility of this class is to generate the files containing features of the packets in order for the kNN attack to use them later on.

The features that are extracted using this feature extraction process are the transmission size feature, unique packet length, transposition, packet distribution, and bursts occurrences. Initially, we create an array list and we add all the features that are extracted from the packets to this array list. After that, the total time and size is added to the array list. Finally, each instance in the features array list is written to the output file that we created to hold the features information.

Fingerprinting Attack

The website fingerprinting attack is done by applying the K-Nearest Neighbour Classifier (kNN), which is designed in a way that can break the defenses. The kNN algorithm is a machine learning algorithm consisting of a group of website instances treated as a training set and another treated as the testing set. Each training point is assigned to a class which denotes the webpage from which the packet sequence is loaded. Given a testing point, the classifier should be able to determine the class (webpage) which the testing point belongs to by comparing the testing points with the points previously used for training. The distance which represents the difference in features between the testing point and the previously classified points from the training set will then determine to which class the testing point will be classified. The point will be classified to the class with the nearest points (least distance). The distance is computed based on the difference between the features of each point. Each feature should have a different impact on the classification process. Some features are more distinctive and give more insight about which class a certain point belongs to. This difference between features is captured by giving different weights to different features.

Learner.java

The Learner.java class will be responsible for applying the kNN algorithm. The kNN attack algorithm is the core of this project. In order to implement this attack, we created the Learner.java class. This class is responsible for learning to classify each packet. The methods in this class will be used to train the algorithm on the training set to reach the optimum weight by which each feature should contribute in the classification process. So far, we have implemented the methods to initialize the weights, compute the distance between two points (packets) to measure how similar they are, and a method to check if an array contains a certain element as a helper method. Our main purpose from this point forward will be to continue the implementation of this algorithm.

Quality and Quantity of work

The coding was done in Java Programming language using netbeans and eclipse softwares by the team members. The defense, the feature extraction, and the attack each was developed in a different java file.

The codes for defense and feature extraction are completed with no compilation errors, while the code for the kNN attack is under development. So far, we have written a total of 490 lines of code excluding the unit tests and 630 lines of code counting the unit tests. In order to ensure the quality of our code and make sure that each method is working in the way we intend it to work, we wrote some unit tests using JUnit. JUnit is the framework of unit testing for Java. All the unit tests are included in the files: FeatureExtractorTest.java, PacketTest.java, and PacketComparatorTest.java. Currently, we made sure that all the unit test run successfully and pass. So far, our unit testing covers only around 15% of the lines of code that we wrote. We are planning to increase this test coverage in the future.

We also did code review before each commit in an attempt to catch any logic bugs or mistakes that the unit tests might miss.

Team member's Contributions

Each member have put in equal contributions in the project. Each member has read the paper and attended the meetings regularly to attain a better understanding of the project. The background research was done by everyone and the understandings was discussed. The technical choice was made based on the familiarity of the language to the team members so that it would be easier to implement. The progress report was compiled by all four of us.

Member's Individual Contribution:

Akshaya Mahadevan	Coding for Feature Extractor and writing progress report
Lina Nouh	Setting up the environment, creating documents google drive for collaboration, coding for Feature Extractor, compilation of coding, and writing progress report
Manisha Mavullapally	Coding for Feature Extractor and writing progress report
Nishanthini Suresh Babu	Coding for Decoy Pages, coding for Feature Extractor, compilation of coding and writing progress report

What's Next?

- Completing the implementation of the kNN algorithm
- Experimenting with the algorithm by applying it to different types and quantities of data sets
- Trying to implement other defenses to test if our attack would still work

- Writing more tests and increasing the test coverage in order to ensure the quality and correctness of the code

Faced Challenges

We faced some challenges while working on this project. One challenge was that some team members were not familiar with the use of GitHub. We spent some time trying to familiarize ourselves with it and learning how to use it in order to collaborate on the project.

Another challenge that we faced was finding time to meet to discuss the project. Since we are all busy masters students and we are taking other classes, it is time consuming to work and prepare for all our classes. We had to find some time that is convenient for all team members in order to set meetings.

Potential Risks

The major risk we are facing is the limited time we have to complete the project. We will try to implement as much as possible from the attack. We also want to try to implement the different defense strategies that were mentioned in the paper. We will implement as much as the time given to complete this project allows.