

Distributed Typosquatting Detector

Distributed Typosquatting Detector is the distributed application with the simple web interface that receives a user input, scans domain squatting URLs in multiple remote machines and dynamically returns back a scan result.

Getting Started

This project is divided into the server and client program.

Running the Server

Requirements

- Java Development Kit 11 or higher
- Apache Tomcat 9.0 or higher

Instructions

1. Clone this repository to the location where you wish to run the server program. Since this is a private repository at the moment, you will have to use the special command as follows.

```
git clone https://your_user_name_here@github.com/msukmoon/typosquatting-detector
```

2. Copy and paste typosquatting-detector.war from the project directory to the webapps directory under your Tomcat home directory.

Note: You may need to use `sudo` to access the Tomcat subdirectories.

3. In the Tomcat directory, run `bin/catalina.sh run` to start the server. Enter the IP address to bind the server to.

Note: It is recommended that you start Tomcat from the command line and not as a service because the server requires input from the console to start up.

4. Go to localhost:8080/typosquatting-detector/search in a web browser. You should see the dashboard.
5. Enter a URL to search for typo domains and click Search or press enter.
6. To shut down the server, press Ctrl-C in the terminal where you started Tomcat.

Running the Clients

Note: You will have to **start the server first** before running the clients. You may run the client program in multiple machines.

Requirements

- Java Development Kit 11 (13 for executable jar) or higher
- Google Chrome
- [ChromeDriver](#) that matches the version of your Chrome
- Xvfb (only for Linux environments)

Instructions for Running the Executable JAR

You could simply run the executable jar for the client side program. Note that you will need Java 13 or higher to run this.

```
java -jar typosquatting-detector-client.jar
```

Instructions for Compiling and Running Source Codes

1. Clone this repository to the location where you wish to run the client program. Since this is a private repository at the moment, you will have to use the special command as follows.


```
git clone https://your_user_name_here@github.com/msukmoon/typosquatting-detector
```

2. Go into the cloned directory.

```
cd typosquatting-detector
```

3. Compile `Client.java`, `ClientImpl.java` and `Server.java` with their dependent jar files in the lib directory.

```
javac -cp lib/\* src/typosquatting_detector/Client.java src/typosquatting_detector
```



4. Run `ClientImpl.class`.

```
java -cp src:lib/\* typosquatting_detector.ClientImpl
```

5. The program will ask for your local IP address, hosted server's IP address and chrome driver path. To run the server and clients within the localhost, enter all IP addresses as `127.0.0.1`. The chrome driver path is usually located at `/usr/local/bin/chromedriver`.

Virtual Machine for Running the Client

The custom virtual machine appliance that is setup for running the client program is available [here](#). It is Ubuntu 18.04.3 LTS that includes OpenJDK 11, Chrome, Chrome Web Driver for Selenium and Xvfb. The password is `1234`. The network adapter for the virtual machine should be attached to `Bridged Adapter` but `NAT` or `Host-Only Adapter`. It is recommended to import and run this appliance in VirtualBox. You may also choose to run the client side program on AmazonVM or Docker.

Program Architecture and Description

Our program uses Java Remote Method Invocation(RMI) to communicate between the Master Node and the Worker Nodes. We used RMI because it is useful in creating distributed systems in Java. Our program also used servlets and Java Server Pages(JSP) for the Web Dashboard. We decided to use servlets and JSP so we could do the entire project in 1 language(Java).

Server Program

The server as a whole functions as the web dashboard and the "Master Node". It is responsible for displaying the form and the results to the web dashboard, as well as taking user input and generating all typo squatting variants of a given URL by using the remote clients.

Servlet

This is a part of the web dashboard. Initially, `index.jsp` contains a form where a user can enter a URL. When the user submits a request, `Servlet` transfers a user input to `ServerImpl` and then makes it to assign work to the clients. Once the clients are done scanning URLs and the server is done assembling the results received from the clients, the servlet displays the results back to the web dashboard.

ServletListener

This contains the `contextInitialized` method that is called when the server side program is first started. It initializes `ClientImpl` and `RemoteGenerator`.

Server

This is an interface which is implemented by `ServerImpl`. This interface is required for remotely invoking server side methods from the clients. It declares the methods that need to be implemented by the server in order to handle client connections, assigning URLs to clients, and receiving results back from clients.

ServerImpl

This is the implementation of the server side program. It first receives a URL, generates typo squatting variants of that URL and then places them all in a `urlQueue`. Then, it assigns one URL at a time to any running clients to be crawled and saves any results it receives back from the clients. It will then display the results on the Web Dashboard.

AdjacentKeys

This has one public method, which returns the `map`, where each key is mapped to an array of keys which it is adjacent to on the keyboard. This `map` is used for generating character replacement typos and character insertion typos.

ReportGenerator

This collects the reports received from the clients. Each of the reports consists of a URL, base64 encoded

screenshot and source code. This reads the reports from a folder, decodes the screenshots and saves them into another folder. It then assembles all the alive URL variants, along with corresponding screenshot and a source code, and creates a single HTML file, which is `report.html`.

Client Program

The client as a whole serves as the "Worker Node". When run, it connects to the server, reports itself by registering to the server and then waits to be assigned a URL to crawl. Once it receives a URL from the server, it uses headless chrome to check if that URL exists, and if it does, take a screenshot and collect the HTML script, then report all of this back to the server.

Client

This is an interface which is implemented by `ClientImpl`. This interface is required for remotely invoking client side methods from the server. It declares the `crawl` method, which the `Client` must implement in order to crawl URLs.

ClientImpl

This is the implementation of the client side program that reports itself for duty to the server by generating its unique ID and then registering itself to the `clientMap` of the server. It then waits for the server to begin crawling URLs from the server's `urlQueue`. Once it receives the URL, it checks if the page exists. If the typosquatting domain is alive, it crawls the URL using headless chrome, collects the HTML script, and takes a screenshot of the page. Finally, it will report the HTML script and screenshot (encoded in base64) back to the Master Node in .txt format, which corresponds to a single file containing image and page source. Concatenating the elements in a single file is also aimed to facilitate the management of files in both server and client sides.

Third-Party Resources

We used Apache Tomcat to run our Server. We also used ChromeDriver so our Worker Nodes could crawl the typo squatting domains. For communicating between the Master and Worker nodes, we used Java RMI.

Authors

We are team 'Unnamed' at Stony Brook University's Fall 2019 CSE 331 class.

- **Henry Crain** - [henrycrain](#) - henry.crain@stonybrook.edu
 - Wrote algorithm for generating typos using the typo-generation model #4 from the Section 3.1 of this [paper](#)
 - Created architecture for managing worker nodes
 - Contributed to design of RMI architecture
 - Added report to web dashboard
- **Hye-Jun Jeong** - [HyejunJeong](#) - hye-jun.jeong@stonybrook.edu
 - Wrote algorithms for generating typos using the typo-generation models #1 and #2 from the Section 3.1 of this [paper](#)

- Crawled typo URLs and get a page source and a screenshot.
- Transferred resulting images and HTML scripts from the clients to the server.
- Generated old and new results as `report.html` to the web dashboard.
- **Myungsuk (Jay) Moon** - [msukmoon](#) - myungsuk.moon@stonybrook.edu
 - Wrote algorithm for generating typos using the typo-generation model #3 from the Section 3.1 of this [paper](#)
 - Made the web interface by using the JavaServer Pages (JSP). Received the user input and then dynamically returned the result.
 - Made the distributed system by using Java Remote Method Invocation (Java RMI). Registered and deregistered clients using their uniquely generated IDs on the server. Made the queue of URLs on the server to be remotely accessed by multiple clients.
 - Set up the VirtualBox appliance for running the clients.
- **Nicholas Reimer** - [nreimer](#) - nicholas.reimer@stonybrook.edu
 - Wrote algorithm for generating typos using the typo-generation model #5 from the Section 3.1 of this [paper](#)
 - Created a way for the server to assign URLs to all running clients concurrently.
 - Worked on creating a connection between server and client(Master Node and Worker Node) with Java RMI.
 - Created the AdjacencyKeys.java file, which generates the map of adjacent keys.
 - Although this did not make it into the final version of our project, connected the server to MongoDB to save results from the Worker Nodes(Instead results from Worker Nodes were saved in files by the server).