# Underground Scraper

## 3 Collection Methodology

Now we turn to describing how we collected the data including underground marketplaces as well as social networks like Telegram. First, we will briefly introduce the main idea and methods, and our achievement measure, and then we will write down some troubles and challenges we met and how we addressed or bypassed them.

### 3.1 Results Overview

We mainly dive into 3 different marketplaces (AccsMarket, EZKIFY Services, BlackHatWorld) and 1 social media (Telegram), yielding millions of rows (tuples), with a $\sim$100 GB data set size. The scraper runs one to two times every day to keep track of various data indicators.

The whole scraper project is written in pure Rust and benefits from its excellent error handling and recovery, to keep it safely running in the background. We have successively used many techniques including Tors[1], IP pools, Chrome drivers, and Headless Chrome (puppeteer) during the whole process.

As of mid-May, the number of tuples (rows) in each marketplace is shown in Table 1.

| Marketplace | Tuples |
|---|---|
| AccsMarket | 63786 |
| EZKIFY | 164 672 |
| BlackHatWorld | 581 103 |
| Telegram | 268 678 097 |

Table 1: # of tuples in each marketplace

### 3.2 Scraping Mechanism

Since viewing posts and contents in many forums of BlackHatWorld does not require authentication, we can do our scraping sneakingly. To keep our efficiency and not be noticed by Website administrators, we use proxy IP pools with 100 different IPs with each one walking at a random 2.5 s $\sim$ 3 s/page rate, and we finally get a quite good rate of 30 $\sim$ 40 pages per second.

Also, BlackHatWorld built a defense of Cloudflare, to resolve it we use Chrome driver and puppeteers to temporarily open a Chrome, and manually solve the annoying puzzle to get Cookie, then the Cookie will store to local automatically, and then this authentication cookie will remain valid for at least 2 $\sim$ 3 days, which offers a lot.

To coordinate all "workers" that use different IPs, we use a local server technique —— We built a local server to handle the functionality of result uploading, it plays a role as a gateway, which means that we can write different kinds of scrapers (in different ways), and all of the data will send to our local server in a simple (and unified) format. These scrapers can written in various languages and we avoid sticking lower into databases. The server also has functions like "load balancing", different scrapers (workers) first request to it to get the disjoint work, and do them themselves, finally upload to the server, thus ensuring the efficient use of resources.

We use PostgreSQL as the data storage and management system for its various features including JSON storage and a beautifully efficient speed.

#### 3.2.1 Data freshness check and update

For AccsMarket and EZKIFY, the website itself does not keep the history of data, we can scrape it regularly by checking and comparing (post-processing), because one round of whole-scraping does not cost much. For BlackHatWorld, we can force the post order by the "post date" (a.k.a. ID order, which is some) by using `?order=post_date&direction=desc`, to prevent the out-of-order caused by irregular replies.

### 3.2.2 Snowball Scraping

For Telegram, we use the technique of snowball scraping. Suppose we already have a set of channels in our database, we can call the "content" subcommand to fetch all the history messages of channels (which are visible to us), then analyze all possible forms of telegram links to explore new channels. By using this method we can get more than 500 thousand channels/groups.

### 3.2.3 Parsing

Our Methodology is to parse in local. We use Rust's "scraper" crate to do modern and fast HTML parsing, and use "`serde_json`" to do structured JSON analysis, finally store to PostgreSQL, which also supports JSON.

## 3.3 Challenges

### 3.3.1 Verification/Authentication

Nowadays many pages are defends with Cloudflare. So we will open a Chrome as a helper role when it can't pass the initial verification, and once the user solves the annoying puzzle we will immediately record the request parameters like cookies by Chrome DevTools Protocol, and use these data to simulate the following requests.

### 3.3.2 TLS Fingerprints

In recent years, Cloudflare set up a defense mechanism namely "TLS fingerprints"[2]. It takes advantage of that different browsers/clients visiting webpages through HTTP will cause different data transportation in TLS handshaking stage and header performance, thereby detecting and rejecting some malicious requests. So we use uniform HTTP/2 (which is case-insensitive and order-insensitive) and uniform ALPN (by native OpenSSL) to bypass this limit.

### 3.3.3 Error Handling

In the scraping process, it's very common to have some unexpected errors (like the page format is wrong, you have no permission to view some posts, etc.). Since our scraper should be stably run in the background like a server, a reliable error-handling mechanism is required. Rather than common language's "try" blocks, we use Rust's `Result<>` type and "tracing" crate to handle errors and record the log gracefully.

## References

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, USA, 2004. USENIX Association.

[2] Zlatina Gancheva. TLS fingerprinting techniques. In Georg Carle, Stephan Günther, and Benedikt Jaeger, editors, *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Winter Semester 2019/2020*, volume NET-2020-04-1 of *Network Architectures and Services (NET)*, pages 15–20, Munich, Germany, April 2020. Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich.