# Identifying and Blocking tracking URLS on web pages

Deniz Aytemiz
Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg VA, USA
denizaytemiz@vt.edu

Azad Fazal
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg VA, USA
azad@vt.edu

**Abstract:**

In the internet world, users are aggressively tracked to manipulate the advertisements. Tracker/advertisers employ tracking code in order to create behavioral profile of the user and employ it to show advertisements. This is ongoing arms-race where trackers try to circumvent the content blocker whereas advertisers try to deploy tracking code by evading these blockers. A tracker is piece of code on the website itself that accesses you and then sends that information elsewhere. Actively visiting the website triggers the tracker to collect any data it can find on you. You can think of it as automatic scanner that passively profiles you. We have built a chrome extension that captures the HTTP requests on the fly before they are sent and analyses them based on the URL filter lists and rule deciding factors to block them.

**Problem Statement and Goal:**

We are all being tracked every time we do anything online. Essentially every online click is being recorded and not just recorded by the company that you interacted with. The creepiest part of it is that being recorded by thousands of so called third parties online. By first party, we mean the website that we think we are visiting, and a third party is any entity that is not on that website and is typically not visible. You are not aware of your interaction with them. Privacy scandals like Cambridge Analytica scandal has shown the detailed profiling that the data brokers collect from our online surfing habits come with serious risks.[1]
Online advertising has never been more invasive and more unescapable thanks to the wealth of data that is available on each user. A single website or and application might contain dozens of trackers building a detailed profile of who you are and what you do online. The bad thing is, we do not even notice. Privacy is a buzz word for all the big tech companies these days. For example, Apple, Google, Facebook etc. have all recently taken vows to undertake new measures to protect user data. How valid those claims are still needs to be seen.
Our goal is to minimize web tracking by blocking the URLs on the fly using the filter list and chrome extension.

**Approach:**
We are going to block the tracking requests on the webpages at the code level. First, we will crawled web pages using available filter lists using chrome extension. Against each request we will have a call_stack. We will use that call_stack information to identify the faulty requests. Once we have the labelled requests, we will use this data to block the tracking URL on the web pages.

**Threat Model:**

The most common tools to collect information about us are so called cookies. Cookies are tiny files placed on your computer by your browser when you visit a website. There are two basic types. Session Cookies and persistent cookies. Session cookies are good. They enable us to move from one part of a website (For Example: a product page) to another (For Example: a check-out page) without having to re-login for every page. Session cookie only lasts as long as your current session and should automatically be deleted when you log off your website. Persistent cookies can be good or bad depending on how you look at it. They are placed in your computer, and they stay in your computer. They are primarily used by marketing firms to track your browsing history. For Example: If you keep visiting a shoe store website, you will soon find ads for shoes following you wherever you go regardless of which website you are visiting. In some cases, this can be intrusive while in others the relevancy of the ads may be welcomed.
Another method is browser fingerprinting. When you visit a website, the website's server has the ability to transfer a snippet of Java Script to the browser. Your browser then executes that script locally on your device. The code then can collect information about installed extensions, bowser types and names, time zones, screen resolution, adblock presence, list of available fonts, computer hardware and many more. Java Script creates a hash of the collected data. This is called your browser's fingerprint and then sends it back to the web server of that website where it is typically stored in a database. Provided that the fingerprint is unique to you, you can be tracked when you re-enter the website. You can also be tracked across multiple websites if they share your fingerprint. This is advantageous for the websites since they do not need to create and store cookies to save data on you. For this reason, the browser fingerprints are also called cookie less monsters.
Of course, many people brush off privacy worries saying they have nothing to hide. But have a look at what information is collected about us: Gender, age, personal status, address and current location, what music we listen to, which TV series we like, where we buy online and what. Individual profiles are created using this data. This allows pretty precise conclusions about personal situations to be drawn. For Example: monitoring your behaviors and purchasing powers. The worst thing about these profiles is that they are worth

a lot of money, and they are being sold. Especially interested in our user data are the bank, landlords and insurance companies. It helps them to access how reliably a customer might pay and the marketing industry uses the data to generate personalized ads.

**Specific work Performed:**

In our chrome extension we have leveraged chrome's web request API and chrome's devTools protocol. DevTools protocol is a set of APIs that allows us to programmatically access the internals of devTools in order to exploit them for our own purposes.[3] Specifically, we have used Network.requestWillBeSent event which is fired when page is about to send HTTP request. When we load a web page, its HTTP request has a life cycle. We can intercept a HTTP request during each step of the cycle using the chrome's web request API. The web request API defines a set of events that follow the life cycle of a web request. You can use these events to observe and analyze traffic. Certain synchronous events will allow you to intercept, block, or modify a request.

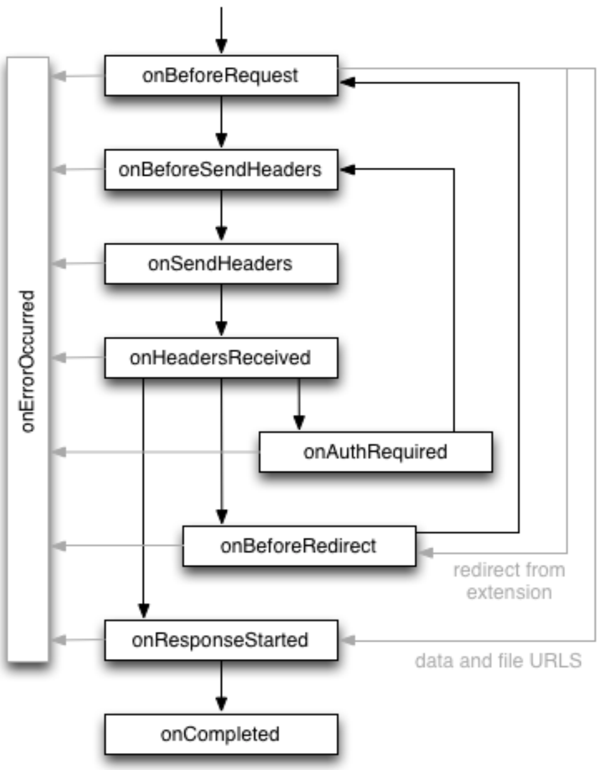


Figure 1: Event life cycle for successful requests

We added an event listener that listens to an onBeforeRequest event to intercept the HTTP request, which is before we see anything on the webpage. This makes it is a good place to listen if you want to cancel or redirect the request.
To cancel or redirect the request, we first include "blocking" in the extraInfoSpec array argument to addListener(). Then, in the listener function, return a BlockingResponse object, setting the appropriate property: To cancel the request, we include a property cancel with the value true. For this to make it work on every website, we used EasyList since it is the primary list of ad-blocking filters that the majority of ad blockers use. We parsed and labeled the easyList using python's adblock parser. We used four EasyList rule deciding factors. Its top level URL, what is its resource type, whether the rule is third party or not and its domain. All four of these factors are used to decide whether to block a particular HTTP request or not.

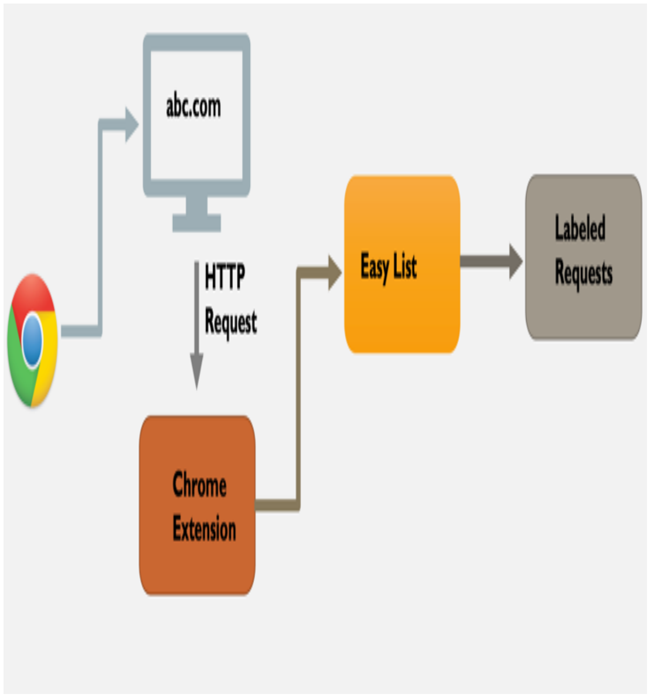


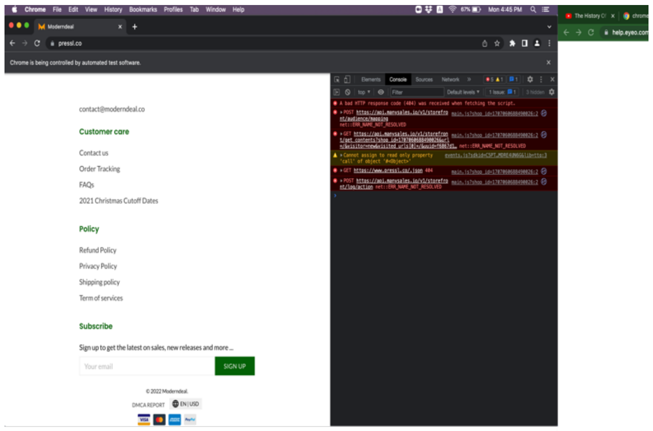
Figure 2: Web Crawling, labelling and blocking methodology

Following are the highlights of our work in order to block tracking requests on the fly:

- We have built a google chrome extension with the help of google chrome's API [2] and capturing the HTTP request on the fly before it actually gets processed by chrome.
- We have used Selenium in python to automate our chrome extension.
- We have used EasyList filter list to block unwanted network requests.
- We have written scripts to parse our filter rules and URL matching.

**Findings and Implications:**

We used selenium in Python to run our extension and observe the blocking behavior. We randomly selected websites from a list of from Alexa top 1million web pages. Most of the web pages were loaded normally and the tracking requests were blocked as shown in the screenshot below. A few pages crashed as well while running the extension through selenium. We are trying to figure out the cause of crashed pages.



```
A bad HTTP response code (404) was received when fetching the script.
▸POST https://api.manysales.io/v1/storefro   main.js?shop_id=17070606884900026:2
nt/audience/mapping
net::ERR_NAME_NOT_RESOLVED
▸GET https://api.manysales.io/v1/storefron   main.js?shop_id=17070606884900026:2
t/get_contents?shop_id=17070606884900026&url
=/&visitor=new&visited_urls[0]=/&uuid=f6867d1… net::ERR_NAME_NOT_RESOLVED
▸Cannot assign to read only property          events.js?sdkid=C5PT…MDRE4UN6G&lib=ttq:3
'call' of object '#<Object>'
▸GET https://www.pressl.co/.json 404         main.js?shop_id=17070606884900026:2
▸POST https://api.manysales.io/v1/storefro   main.js?shop_id=17070606884900026:2
nt/log/action net::ERR_NAME_NOT_RESOLVED
```

Figure 3: A webpage loaded by selenium and blocked requests

## Conclusion and Future Directions:

We have successfully blocked HTTP requests by building a chrome extension that leverages EasyList to identify faulty URLs. Moreover, scripting using python's ad block filter parser API enabled us to determine blocking rules for labeling the methods that as blocking or non-blocking. A future research direction in our work would be to use machine learning models to circumvent the need of manually updating filter lists.

## References:

[1]
https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal
[2]
https://developer.chrome.com/docs/extensions/reference/webRequest/
[3]
https://chromedevtools.github.io/devtool/protocol/tot/Network/
[4]
https://adblockplus.org/filter-cheatsheet#blocking