Imperial College
London

INDIVIDUAL PROJECT FINAL REPORT

DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Quantifying the Privacy Techniques of Adblockers

Author:
Dylan Tracey

Supervisor:
Arthur Gervais

July 1, 2018

Submitted in partial fulfillment of the requirements for the Type of degree of
Imperial College London

Abstract

In this project, I have created a website which can benchmark browser configurations (including extensions) against various third party and first party tracking techniques. Research has been done focusing on quantifying adblockers by numerically assessing their ability to block third party domains across existing popular websites[1]. However, in this project my focus will be to isolate different tracking techniques and see if adblockers can algorithmically detect tracking rather then the usual method of relying on a blacklist. In particular, I focus on testing Privacy Badger. Through this approach, I have found that although traditional third party cookies and HTML5 localStorage super cookies are blocked by Privacy Badger, weaknesses in the cookie deletion heuristics mean that tracking can be done. I therefore have concluded that you cannot algorithmically determine which cookies are tracking or not unless by utilizing a blacklist. This is primarily due to easy to implement host-slave domain networks that can appear low entropy to adblockers (such as Privacy Badger), but deceptively track the user. This can then be used to create a history log of the users browsing habits across the first party domains that use these malicious host-slave domain networks.

Acknowledgments

I would like to extend my gratitude to Arthur Gervais. He was a great supervisor and his suggestions and guidance helped me throughout the project.

Contents

Chapter 1

Introduction

In this project I have designed a deliberately malicious group of websites, which can simulate different methods of tracking users. This can the then, in browser, benchmark how well different adblockers can identify these threats. In particular, this benchmark shows the benefits and weaknesses of a popular privacy focused adblocker called Privacy Badger[2].

1.1 Privacy Concerns with Advertisements

With the advent of the Internet, and the continual accelerating pace of technological progress, society is changing at an ever increasing rate. In a short time span, our lives have been completely revolutionized. From on-line shopping, to new ways to consume news, and completely different methods to communicate with friends via social networks - the way we navigate our daily lives has changed dramatically. However, with these great technological advances, there are not only benefits but also potential and realized drawbacks.

For example, in the desire to quickly disseminate breaking news, search engines can be exploited to suggest fake and unsubstantiated stories. Another example would be the particular concern for those working in the exciting arena of Artificial Intelligence, that a general AI could in the future have conflict with the goals and desires of humans.

However, for the topic of this project, I will be addressing a very real threat introduced to our lives with modern on-line advertisements. Although seemingly innocuous, as to the layman all advertisers are doing are moving the posters we see on the streets and videos we watch on the tv to the on-line space - the ramifications of on-line website advertising can be intrusive to our privacy. Due to the particular nature of 3rd party advertising, users navigating the Internet can have their browsing histories mapped out. Although some might deem the intentions of this data collection noble (as advertisers could use this data to target users with content that is

more suited to their tastes), the potential for this data to compromise privacy and be available to the highest bidder is present. Simply put, this pernicious form of tracking reduces the privacy afforded to the individual completely, unlike the traditional poster-on-the-wall type advertisements typical in the pre-Internet era.

Fortunately, a fairly common method exists that can reduce/hide or eliminate the presence of advertisements when browsing the web; Adblockers. These can be added quite easily as an extension to your browser. Some popular implementations include the titular Adblock[3] and Ublock[4]. However, while these popular adblockers might improve the user experience of someone browsing the web, they are not geared towards improving privacy algorithmically first and foremost. To be precise, the usual reason to download these types of adblockers is so you don't have annoying pop-ups when visiting websites and so you can skip youtube video adverts. Privacy is not usually the concern to most users downloading these adblockers.

Thankfully, an adblocker which addresses these privacy concerns as a core central tenant has been developed and maintained. This adblocker is called Privacy Badger[2]. Originating as a project from the none profit Electronic Frontier Foundation[5] - this open source adblocker focuses purely on the privacy of its users in an algorithmic fashion.

1.2 The Problem

In this project, I will be evaluating and comparing the performance of Privacy Badger in comparison to other adblockers to see the degree to which it improves the privacy of users with respect to 3rd party advertisements. In order to facilitate this analysis, I will focus on developing a set of deliberately malicious websites that can test the adblockers with different techniques to track the user. The motivation behind using a more theoretical approach wherein the techniques are the important grounds to test adblockers and not a very contextual analysis (like testing against alexa's[6] top 500 viewed websites), is because this is the only true way of testing the robustness of the adblockers. If testing were based purely on the real world top website testing form of analysis, adblockers could simply focus on creating blacklists for the top sites visited to seem as if the underlining algorithms are blocking 3rd party trackers, when in reality this is not the case. Instead of the behaviors of the 3rd party trackers being used to determine what should be blocked, the adblockers are simply creating a blacklist that can in the best case only play a cat and mouse game updating the blacklist with 3rd party trackers as new domains and techniques are added.

This is evidently no guarantee of privacy at all and only ensures that some preselected sites will be blocked in a manner that has nothing to do with the effectiveness

of the underlying algorithms but simply a list of domains. This simple fact gives credence to my idea of creating a comprehensive range of deliberately malicious sites which can be continually improved on with more techniques and methods to compromise the users security when found and thus give a benchmark to measure the adblockers blocking capabilities. As will be mentioned later, a simple iframed third party tracking cookie example has already been created and tested as part of Privacy Badger's continuous integration. This provided me inspiration to create a dynamic website version of such simple static tests and to generalize the website to be used to test any adblocker extension via a benchmark.

1.3 Contributions

At first, I looked at using a web driver (like selenium[7]) to develop an application that could test various browsers and adblockers, but instead I decided to streamline and simplify this approach by featuring the benchmarks as a part of the web application. This benchmark goes through the maliciously made websites sequentially and reports the results on whether 3rd party domain calls were able to track you. For instance, a simple iframed third party tracking cookie would be one test of many which would either report "Success" (that a unique identifier was able to be produced as a cookie to track your browsing habit) or a "Failure" (the website was unable to create a unique identifier cookie to track your browsing habits).

Therefore with these websites, this project creates a measure of privacy based on a theoretical rather than a "cat and mouse" approach as described previously, and this should ultimately help improve peoples understanding and evaluation of user tracking techniques. Also, due to the iterative nature that the development of such malicious websites has taken (more techniques can be added when discovered), my project could be built upon and improved with state of the art and upcoming tracking techniques as they are unearthed. Once I am ready to open source this project, the potential for creative ideas and suggestions to exploit the vulnerabilities of surfing the web could advance forward the privacy focused adblocking industry.

In total, I was able to design seven different techniques to track the user. The most popular modes of tracking used today are implemented including First party only cookie tracking (which should be allowed and is expected) and third party cookie tracking (which we should seek to avoid). In addition to these simple tracking approaches, HTML5 local storage super cookies were implemented. Another tracking method I have implemented is the concerning "link shimming" that popular first party websites do track which external links are clicked. Finally, three different ways to circumvent Privacy Badger's low entropy cookie filter are included which calls into question whether we should allow cookies to be set at all. Thus through my

testing of these tracking modes, I have concluded that it would be virtually impossible (without a blacklist approach) to algorithmically determine which cookies are harmful or not.

Chapter 2

Background

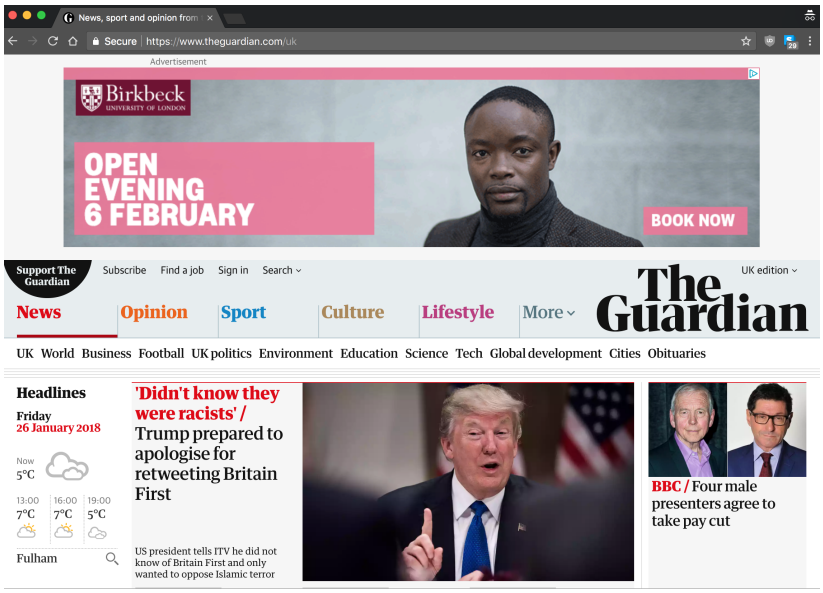
2.1 Types of Advertisements and Cookies

There are a variety of different types of advertising that can take place on a website. One of the least talked about and most simple forms of these advertisements is having a first party advert. Essentially, the website will display either a video or photo advertisement which has been selected by the website/company to directly advertise. The inflexibility of this approach demands man hours of programming to constantly update the adverts as well as direct relationships with each prospective company. This problem means that a lot of websites, particularly with smaller companies, would rather outsource the job of advertising to an external party.

Thus third party advertising services are generally used. These services, like DoubleClick[8] as an example, will have a plethora of adverts to distribute among the 1st party sites that host their platform. This form of advertising has many advantages. Firstly, this middleman approach between advertisers and websites ensures a smoothness of transaction. Advertisers do not need to directly communicate with each of the sites it will be hosted on. All they need to know is that it will be shown to users a certain amount of times. This is a reduced cost for companies looking to advertise a product as they now can streamline or eliminate departments relating to communication with host websites. Similarly, on the other end in which websites are looking to generate revenue from advertising. All the website now needs to know is a certain part of the websites HTML code will have a frame dedicated to displaying these ads. The worrying about implementation is now outsourced, leaving valuable man hours for other parts of the website, reducing costs for the company.

While this seems beneficial to the consenting parties involved in this business transaction, the web surfers face privacy consequences. In an understandable effort to offer advertisers and websites a better service, competing ad serving platforms would want to find ways to be more competitive. And this just so happens to lead to the concept of targeted advertisements. It is simply a waste of an advertising spot if a

Figure 2.1: An example of a DoubleClick advert featured on the Guardian (`www.theguardian.com`, accessed 26/01/18)



web surfer is shown an advert which he/she has no interest in. Therefore ad serving platforms can collect data from users, in particular a users web browsing history. By using cookies which are set within a users browser, a unique identifier can be set which the ad serving platforms can use to cross check which first party websites (by using a referrer) the user is viewing and build up a dedicated history of the websites visited. Through data analysis, this allows these platforms to create a profile for the user, such as interests and probable age ranges. This information can then be used for a multitude of different purposes, like serving more appropriate ads, to selling this information to bidders. Regardless of what this data is used for, privacy of the user is now compromised.

2.2 Tracking Techniques and Goals

2.2.1 Basic Tracking

There are many different ways to uniquely identify users and thus create a history log of their browsing habits. The basis of most of these methods starts with the HTTP referer header tag (whose name is such as it was a misspelling of referrer which gives its very intuitive semantic meaning). Therefore, When a first party website calls a third party domain URL (through an iframe, script or any other means), included in this HTTP request will be the origin website. For example, if a website that has been requested by the user, say `www.example.com`, iframes (an iframe is a html tag which embeds the external website as a frame) in another domain, say `www.thirdpartyexample.com` - the server responding to the request at `www.thirdpartyexample.com` will be able to tell that via the HTTP referer tag that the originating website was `www.example.com`.

Now, all that is needed is to identify the user. Once the third party website knows which specific user made the request, a history log entry can be made. The website would thus know that this user visited this first party website (via HTTP referer). As this third party domain can be called externally across multiple first party websites that include it, a detailed history log of previously visited first party sites can be generated. So what methods are there that can uniquely identify the user?

Intuitively, the IP address received from the request might be able to be used to uniquely identify users. However, there are several limitations with using this as an approach. Firstly, a person might use multiple different routers through the same device. For instance, if you bring a laptop to different locations (work, home, university etc), different IP addresses are used which means this does not uniquely identifier the user as expected. Secondly, the public IP address which a web server will receive on a request only distinguishes between nodes in the network such as a router. It cannot distinguish between different host devices connected to the router. Therefore, this can only identify an individual if one person is using the router which is an unlikely scenario. This is because in the workplace and at home, usually multiple people will use the same router connection thus IP addresses are not enough to identify users. Finally, IP addresses aren't permanent and can be changed by the Internet service provider at anytime. Combined, the aforementioned drawbacks of trying to use IP addresses to identify users make it impractical. Thus a more proactive server side approach is required by the developers of tracking services.

Of course, as mentioned in the previous section, there is cookie tracking. The third party domain can set a cookie which is a piece of information stored in the users browser. Each cookie comprises of multiple different field such as 'expires' (at what date the cookie is invalid), 'path' and 'created'. But by far the most important of these fields are 'name' and 'content'. The 'name' and 'content' fields essentially act as key-value storage. Each domain, including domains called via other domains, can store multiple cookies in the users browser. Consequently, all that is needed it to identify a user is to create a sufficiently long enough string of random numbers or characters and set that as the 'content' field of a cookie along with a universally used string such as 'id' and set that to the 'name' field. If the 'content' field is unique, which it is likely to be if it is long enough (or can be enforced by server), each time the domain is called it can check if a cookie has been set and get the unique identifier from 'content' (or if the cookie hasn't been set, set it as described above). Thus this domain knows which user made the request.

Although traditional cookies are still the most widely used method to track users, changes in our browsing habits create issues with their deployment and necessi-

tate new modern ways to track users. This is due in particular to the popularity of mobile browsing. Some mobile browsers, such as safari, do not allow third party cookies at all. In general, cookies are handled differently on most mobile devices compared to their desktop counterparts. Thus advertisers need new ways to track mobile users, especially as this is now such a huge portion of on-line traffic. Just a couple years ago the guardian reported that mobile browsing globally has overtaken desktop browsing[9]. Fortunately for advertisers, there are other ways to track users.

2.2.2 Advanced Tracking

Different new ways of tracking users are referred to informally as 'super cookies'. Due to the vagueness of this definition, multiple types of tracking that are different to traditional cookie tracking all fall under this umbrella term. Some of these methods include storage on other frameworks utilized by websites such as flash, silverlight or HTML5. As super cookies are used to capture the widest net of users, the predominant web mark up language - HTML5 - is a great option to store super cookie information. This is done by using localStorage methods[10]. This can set the key-value pairs the same way cookies can. However this storage is interacted with client side in the javascript rather than server side like traditional cookies. Therefore, an additional step is required to notify the server after the rendering of a website to retrieve the client side information. Either another request is needed via a redirect, or an AJAX (Asynchronous JavaScript and XML) implementation is required. As shown later in the implementation section of this report, HTML5 super cookie tracking is not as hard as it seems and it is able track mobile users; even with the IOS safari browsers 'Prevent Cross-Site Tracking' setting on.

Fingerprinting is another popular tracking technique. Also based around HTML5, this works by drawing to canvas text with a font, size and background color combination (via HTML5's canvas element)[11]; transforming the pixels into a textual format and hashing this to create the identifier. This works as different devices have different GPU variations. However, this is not guaranteed to be unique, as some users will inevitably have a similar GPU.

Additionally, there is another tracking technique which isn't focused on generating a first party visit history, but rather tracking which external links are clicked. Accordingly, it is a first party site that is doing the tracking rather than a third party. This tracking method is called link shimming. In essence, link shimming involves a website replacing what should be an external destination link to third party content, with a URL which will first process (which can log the link the user has clicked on) before redirecting to the external link the user intended[12]. Therefore, a log of the externally clicked on sites can be generated for the user of a first party site using link

shimming, which is some sense violates a sense of privacy a user might expect. Even more sneaky is the way this is concealed to the layman user of the site. For example, on Facebook, The href field of an external third party link is displayed when hovered over by the user with their mouse (the link looks real), but is replaced by the link shimming URL as soon as the user clicks making the process nontransparent to the user.

However, there are reasons other then tracking why a big website such as Facebook might want to link shim. As addressed in a post[13], there are multiple security reasons to link shim. Using the HTTP referer field, an externally clicked link would get the URL from Facebook which might contain sensitive information related to the users identity. This link shim redirect to a different part of the Facebook website would thus obscure this sensitive information. Also, potentially malicious external links could contain redirects within their URLS which disguise the true destination of the link. This is called an 'open redirector'. Thus Facebook can detect this and warn users that the site might be harmful.

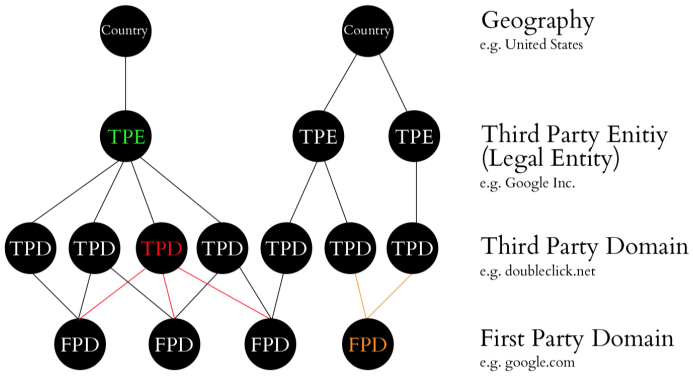
Still, there is a privacy concern nonetheless. As discussed later in the Background section, there are also ways to address concerns that drive Facebook to use link shimming, which are much more privacy conscious.

2.3 Adblocker Techniques and Goals

Adblockers are used to reduce and eliminate the presence of adverts when browsing the web. When coupled as an extension to a users browser, adverts can be blocked or hidden from the screen to increase the experience of the user. The method commonly used is to use a blacklist of domain names of known advertisers and ad serving platforms and then cross reference these to block the relevant 3rd party calls, or hide them so they appear invisible to the user. This approach is used by AdBlock as an example[14], with its blacklist of choice being EasyList[15].

This unfortunately, is a fairly unsophisticated and blunt approach. Blacklists are unsophisticated as it only prevents the current and most popular advertisements. No attempt can be made at all to prevent newer ad platforms until the blacklist can be updated. This means the adblockers are always a step behind and are not really algorithmically addressing the problem, but papering over the cracks as they form - a metaphorical cat and mouse game between advertisers and adblockers. This approach is also blunt, because no attempt is made to distinguish between valid privacy respecting advertisements and nefarious ones. The revenue that should be generated by websites that are respecting the users privacy is reduced when people use adblockers and thus no incentive to ethically advertise is created with this ambivalent form of ad blocking.

Figure 2.2: Visualizing the connection between the first party domains and third party domains. Sourced from 'Quantifying Web Adblocker Privacy'[1]



But why have adblockers generally been developed in this manner? In general it comes from the desires of the user. It is easy to see why users of adblockers would select this form of ad blocking. The less adverts the better irrespective of whether it is ethical or not, and also users can be unaware of how much of their data is being collected and the privacy implications. The former, a human desire is unlikely to change without legislation, but the latter can be improved with awareness from users about how there data is used. News articles or easy methods to test privacy could be ways for users to care more about where they want their data to go.

2.4 Adblocker Privacy Analysis

In order to test the privacy implications of third party advertisements and in general third party domains (distinguishing between these two is inconsequential as far as privacy is concerned). Papers such as the cited[1] "Quantifying Web Adblocker Privacy" has explored a framework to test adblockers and demonstrated a simple way to visualize the process. Indeed as shown on figure 2.2 above which is from the paper, a nodal system is all that is required to have an understanding on which 3rd party websites can track users history on 1st party websites. The third party sites would have knowledge to all first party nodes with incoming edges. So we can simply thus check whether the degree of a third party node with respect to edges from the level below it, is greater than 1. The potential for tracking is thus established.

Of course, you can conceive of tracking that can circumvent this model - if two different third party base domains are owned by the same company, it can then relay information to another site. But this would require a tremendous ownership of unique domain names on part of the company. This is because for each first party

website the tracking company would need to have a unique 3rd party domain name, so tracking over a large sample of browsing history would therefore be costly and inefficient.

The paper does improve knowledge on how ad blockers can reduce privacy concerns by breaking these edges on the graph (in practice by the blacklist blocking the calls from the 1st party website). But this approach of using the Alexa ranking to formulate this analysis, although useful, is slightly flawed. This is due to how adblockers work as described on the previous section. This testing for privacy, only really tests the blacklist and using Alexa will further distort the actual algorithmic capabilities of these adblockers. What we should really be interested in, is the techniques and behaviors exhibited by the adblockers - not whether popular sites have adverts blocked through blacklists. But is it even possible to determine without knowledge of the 3rd parties inner workings whether it can track you or not? This leads us on to a surprisingly effective ad blocking solution which will be discussed in the next half of the background section.

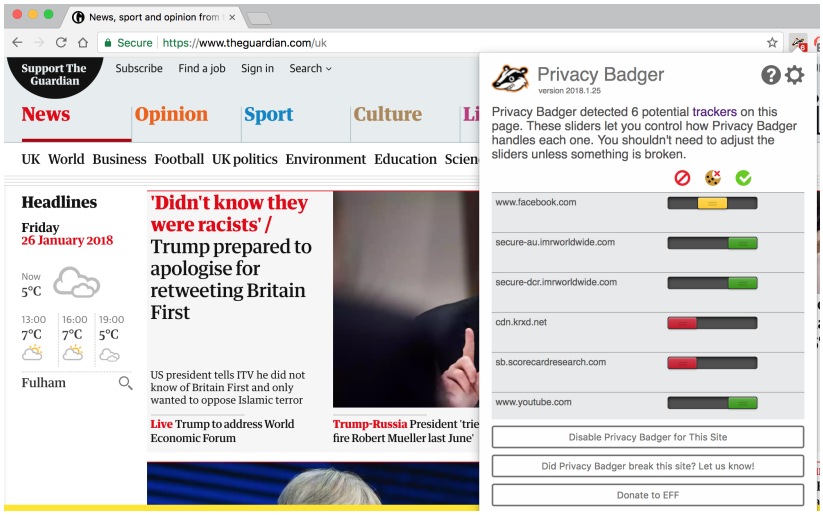
2.5 Privacy Badger and Nodal Analysis

Privacy Badger[2] is an ad blocker project by the EFF (Electronic Frontier Foundation[5]). The unique selling point of this ad blocker in comparison to others is its almost surgical focus on privacy (if it wasn't obvious from the name). EFF, being a non profit advocacy group has an interest in Privacy Badger being ethical, which is evident throughout the development of this adblocker.

The core concept of Privacy badger is that this is an algorithmic approach completely divorced from the context of the web. It doesn't know which domains to block based on based on a predetermined list, but rather the slate is blank to start with and from learnt behavior while a user browses the web, knowledge is accumulated on what to block. Thus the more you browse the more Privacy Badger improves.

So how does this all work? The fundamentals of Privacy Badger is that a global list of 3rd party domains is continually added to. For each website you visit, Privacy Badger adds all the 3rd party domains requested that haven't been seen before to its global list. The states that these 3rd party domains can be classified into is red, amber or green. Red means that this third party domain will never be HTTP requested, it is thus blocked. Amber means that the cookies will be deleted along with the referrers before making the HTTP request to the domain. And finally, green means that the HTTP request is made normally with no interference. It is easy to see the purpose that each of these states will serve when it comes to privacy. Red is used to prevent 3rd party tracking domains. Amber is used to prevent cookies being stored that can identify you, but to allow what might be important third party functionality that

Figure 2.3: Privacy Badger classifying a website into its traffic cone system.



improves the web page. The amber symbol is used when according to the yellowlist of important domains, privacy badger needs to allow the functionality of this third party. And green is used when there is no evidence of tracking at all.

The important final step is therefore how Privacy Badger is able to classify 3rd party domains into the correct categories. And in a pleasing link to the research described in section 2.3, is does so with similarity to the aforementioned nodal fashion. Just as in the ”Quantifying Web Adblocker Privacy” paper, when a node has a degree past a threshold (in the paper it is greater than 1, but Privacy Badger blocks those with greater than 2), the 3rd party could potentially be tracking, and thus Privacy Badger will block the tracking cookie and if not prevent the domain from even being called. Therefore, as soon as three different first party websites have referenced the same 3rd party base domain - it will be put into amber or red categories. Tracking will then in both cases be prevented. Otherwise the 3rd party is left in the green state. Thus the more the web is browsed the more 3rd party domains are added to the global list, and the more first party sites are related to the third party domains - which will steadily increase over time the amount of blocked or cookie blocked requests.

2.6 Cookie Deletion Heuristic

Unfortunately, the proceeding section is somewhat incomplete as there is another way in which 3rd party domains can be considered to be none tracking regardless of whether it is used in 3 or more 1st party websites. This is where a heuristic algorithm comes in which detects whether the cookie the website stores is tracking or not.

In order to create a unique identifier for a person, the cookie needs to store a value of sufficient size. This is obvious through and example, say if there are 10,000 users

of a website, you would need a unique identifier value of at least a 4 digit length (0-9999) to encode them and in practice there are many more users of websites.

With this in mind, Privacy Badger estimates an entropy level for each of the cookies. If a data type is not one of those within a specified list, the cookie is considered tracking, and even if the data type is in the list, if the accumulation of the cookie pairs exceeds the entropy value of 12 - the domain is considered tracking. This ensures that the amount of data that a cookie can store is severely restricted - only three digits, or one digit and a language code would be allowed. Obviously this is simply not enough data to create a unique identifier for the user and therefore cookies which are low entropy cannot be tracking.

This additional check ensures that not all cookies are blocked. If all cookies were blocked once the nodal analysis was greater than 2, this would be bad in terms of website functionality as cookies for specifying languages or other simple and useful functions would be missing.

Although this heuristic seemingly ensures that the cookie cannot track the user, I believe I have found an exploit that is described in the next section of this report.

2.7 Additional Super Cookie Deletion Heuristic

Privacy Badger in a similar matter to normal cookies, uses this entropy level estimation for HTML5 localStorage. Associated with each domain specific localStorage is its size, so if the localStorage for a third party iframe is above a certain amount of bits, there is enough information to potentially be tracking (due to the high entropy). This is restricted to 256 bits. Thus due to the way key-value pairs are stored, using as short a name for the key as possible (one character long), a third party tracker can store a four character long string as a value before being considered potentially tracking. Although this privacy badger heuristic allows more possible combinations than with the traditional cookies heuristic (4 characters vs 3 numbers), this is still too restrictive to allow unique user identification.

2.8 Preventing Link Shimming

Privacy badger also has methods to deal with link shimming, in particular focusing on replacing the link shimmed URL's with the proper external href link. However, this is geared to the specific sites of Facebook and Twitter only. Essentially, privacy badger goes through the page and replaces all link shimmed URL's with the destination URL's that would otherwise be tracked before redirecting. This therefore ensures that Twitter and Facebook cannot log which external sites you click on. As

mentioned before, Facebook might have valid privacy concerns which prompts them to link shim, but privacy badger also takes this into consideration. To do this, Privacy badger ensures that the HTTP request when clicking the external URL has no referer header tag. Therefore, the externally linked site cannot gather private information from the HTTP referer URL. The other concern that prompts Facebook to link shim is the prospect that the website being linked is malicious and might contain open redirectors. Privacy Badger does not address this, but assumes that the user will be careful about which site they are on and intends to view the corresponding site. By offloading this responsibility towards the user, this can lead to potential malicious website being clicked on without a warning prompt by Facebook; nonetheless, this might a risk worth taking, especially as security conscious users would know to not trust external links and would rather benefit from the privacy protection from Facebook and Twitter tracking.

Chapter 3

Design

3.1 Application Beginnings

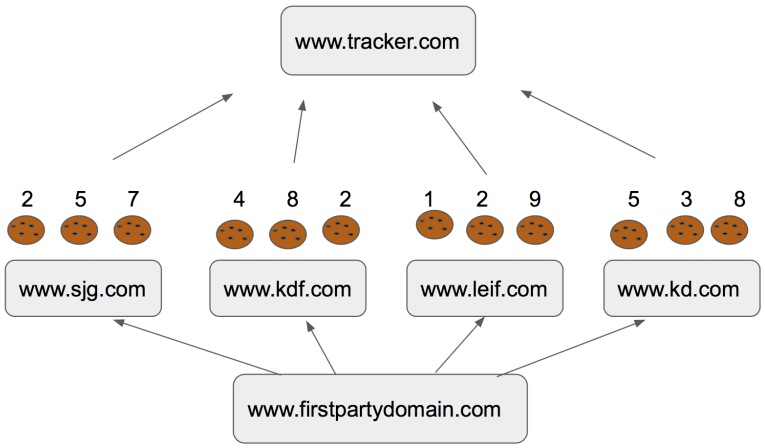
3.1.1 Initial Ideas

To kick off the application portion of the project, I started to find potential weaknesses in Privacy Badger's approach to prevent tracking. Upon examination of the code as described in the background section, I came up with an approach which can be used to uniquely identify users while still remaining undetected by Privacy Badger.

This idea (which would comprise of a technique to add to my benchmark software) that I discovered was a clever way to circumvent the cookie stripping heuristic algorithm used by Privacy Badger. Privacy Badger recognizes some cookies are required by 3rd party domain calls for the functionality of the website. It thus contains a heuristic algorithm as explained in the background section that can classify cookies by how much information they reveal about the user. Typically, this means only a tiny cookie is allowed with a few values (only around 3 value pairs of numbers between 0-10 can be assigned or a single language code).

This ensures creating a unique identifier for a user is impossible (at least if the number of users of the site is large as you can obviously identify 100s of users with a few 0-10 values as a unique identifier). Or is creating a unique identifier impossible? As it turns out I have found a vulnerability in this approach as shown in the figure below. The way it works is having a sequence of separate 3rd party base domain calls on your 1st party website. Although seemingly unrelated to each other, when the first party site is loaded by the user, these 3rd party calls are considered separate and thus the cookies on each are considered separately. If we then ensure the cookies on these 3rd parties are of low entropy and have small values we can encode a unique identifier by splitting the digits up over the 3rd party cookies. These 3rd party websites then relay their part of the identifier to a host website which is now

Figure 3.1: My idea of creating multiple cookie with low entropy values to avoid detection that can be appended to create a unique identifier.



twice removed from the first party domain.

For instance, a unique identifier of 17246745 (shorter for the sake of explanation) which is too high entropy to be stored as one cookie undetected by Privacy Badger. Could be split into chunks: 17, 24, 67 and 45. These are then stored as separate cookies on each of the ”slave” 3rd party domains and relayed to a ”host” website. The host website can then, if the time period is small between the messages from these 3rd party domains, can construct the unique identifier by appending these chunks. Now the host website has knowledge of the first party website you visited.

This approach of splitting up the unique identifier into multiple separate smaller identifiers to be combined also applies to the super cookie portion of Privacy Badger's tracking prevention (as described in background section 2.7). Privacy Badger allows 256 bits of local storage information per domain. Therefore, simply splitting the unique identifier into multiple parts and putting them in separate third party domains is doable in the same way.

I have considered ways that Privacy Badger can be updated to counteract this, but the only solutions I can see is prohibiting all third party cookies - this is not ideal due to reduced website functionality. If we try and actively stop this behavior, as we cannot tell which 3rd party domains are related to one another without a blacklist of some sort, we would have to sum over all third party cookies and if a certain entropy is reached - ban all cookies from being set for this page. Therefore, even the cookies that have nothing to do with tracking get deleted in the crossfire. Also as described

in the introduction (The Problem), if blacklists are the only solution to counteract the tracking, this is the worst case scenario. We are then reduced to a cat and mouse game and are not preventing behaviors we deem unacceptable.

3.1.2 Initial Prototype

To test the idea of splitting the unique identifier I created a prototype that works locally on my computer without the need to write server side software. The process of making the prototype was expedited by the helpful use of a continuous integration test located in Privacy Badgers code (in a pull request)[16]. This test case provided the HTML and JavaScript for a single cookie. All that was needed was to change the first party HTML to instead of loading a single iframe with the third party domain, loading multiple iframes which each stored smaller cookies (3 digits to avoid Privacy Badger's tracking cookie detection). The third party Javascript was thus simply changed as follows:

```
function updateCookie( ) {
    var oldcookie = document.cookie
    var val = Math.floor(Math.random() * 9);
    console.log("read cookie: " + oldcookie);
    document.cookie = "thirdpartytest=" + encodeURIComponent( val );
    console.log("updating cookie to:" + document.cookie);
}
```

Therefore the random value used to encode the cookie is only one digit long instead of the long length that would be blocked by privacy badger. Then instead of just using one iframe instance of the third party HTML with this javascript, I made the first party HTML load in four different third party instances as shown in the file `split_cookies_2` below:

```
<!DOCTYPE html>
<html>
    <head>
        <script src=first-party.js></script>
        </head>
    <body>
        <p>Welcome to the cookie tracker test site.  I'm creating a
        ↪ localcookie for this domain.
        </p>
        <button id="newwindowbutton" class="button"
        ↪ onClick="window.open();">Click this button to open a new
        ↪ window</button>
```

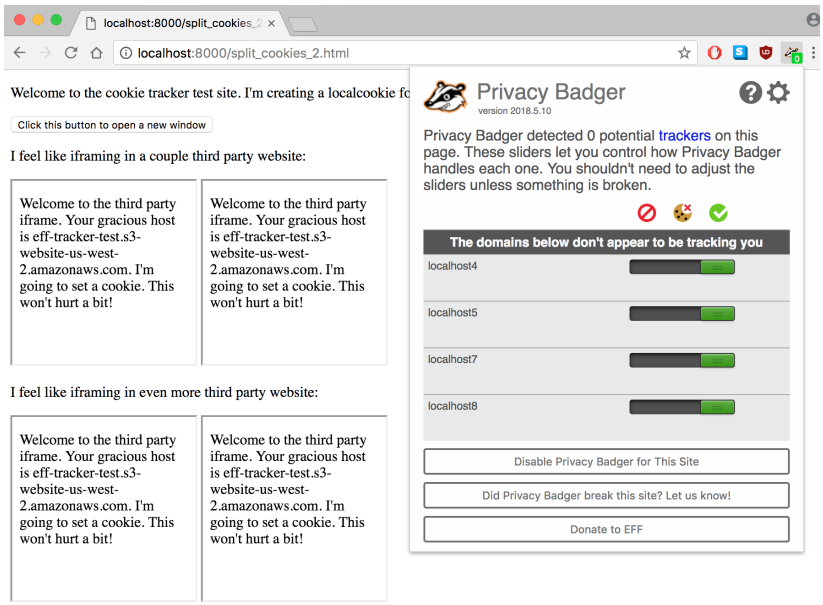
```
10          <p>I feel like iframing in a couple third party website:</p>
11          <iframe src="http://localhost4:8000/external_server/
12          split_cookies_third_party.html" width="200"
    ↪   height="200">where's my iframe?</iframe>
13          <iframe src="http://localhost5:8000/external_server/
14          split_cookies_third_party.html" width="200"
    ↪   height="200">where's my iframe?</iframe>
15          <p>I feel like iframing in even more third party website:</p>
16          <iframe src="http://localhost7:8000/external_server/
17          split_cookies_third_party.html" width="200"
    ↪   height="200">where's my iframe?</iframe>
18          <iframe src="http://localhost8:8000/external_server/
19          split_cookies_third_party.html" width="200"
    ↪   height="200">where's my iframe?</iframe>
20          </body>
21  </html>
```

Due to the need to use different URL's to simulate the four different iframe's as if they were separate domains, I then edited my macintosh's `private/etc/hosts` file which maps domain names to IP addresses. For instance, `127.0.0.1` (the local server) is already mapped to `localhost`. Therefore a URL such as `http://localhost:8000/example.html` would function the same as `http://127.0.0.1:8000/example.html`. It was then rather simple to edit this file to create different domains that the browser would treat separately (although they pointed to the same local server). I simply added the mappings to the hosts file as shown below:

```
127.0.0.1       localhost
127.0.0.1       localhost2
127.0.0.1       localhost3
127.0.0.1       localhost4
127.0.0.1       localhost5
127.0.0.1       localhost8
127.0.0.1       localhost7
127.0.0.1       localhost9
```

Now, I could load the first party HTML as aforementioned, from different hosts to simulate different first party domains having the same four third party iframes. I would simply change the base domain. I would visit `http://localhost:8000/split_cookies_2` first followed by, `http://localhost2:8000/split_cookies_2` then, `http://localhost3:8000/split_cookies_2` before finally going back to `http://localhost:8000/split_cookies_2` to see if the original site was blocking the iframes from loading due to the detection of cookies. To do this I ran a simple local server on my computer using the command `php -S 0.0.0.0:8000` and I was therefore able to view my local HTML files. When using the original first party HTML that only iframed in one long cookie, after visiting these three sites Privacy Badger would

Figure 3.2: When locally testing splitting unique identifiers into substrings between separate cookies, Privacy badger allows the cookies as they are considered low entropy



block the third party from setting a cookie. But when using the `split_cookies_2` HTML and browsing the three different sites - Privacy Badger did not block the cookies (as in figure 3.2). As I believed was the case, the cookies were considered from separate domains (`localhost4`, `localhost5`, `localhost7`, `localhost8`) and thus as they were low entropy cookies of single digits - Privacy Badger did not block the third party iframe when returning to the first URL.

This was the confirmation needed that this idea would work in practice and all that was needed was to implement this as a proper website. Then these separate iframes could send this information to the master third party server which would piece together the separate portions of the cookies and append them together to create the unique identifier. Finally, I was now ready to move from client side experimentation, to a concrete client AND server side implementation. I would then add this splitting approach as a potential technique on the benchmark which could trick Privacy Badger. All I needed was to start this benchmarking website.

3.2 Development Stack

3.2.1 The Technology

The core component of the web application is developed in the web framework Flask[17]. Being a framework for python, Flask was easy to use due my programming experience in the language. I previously had familiarity with Django[18] as a python based web framework, but wanted to try Flask due to its simplicity of design and flexibility. Also, it seemed more suited than Django to develop an individual application due to there being less boilerplate code and greater customization in terms of choosing how to structure your web application. This greater choice did mean more time needed to be spent on design considerations - such as considering how to implement databases and which third party libraries to use - however, this helped me only use the features that were necessary. In Django, much of the boilerplate code and way of developing the application would not be required.

To test and develop my application locally I used the IDE (Integrated development environment) PyCharm[19]. This helped check for errors, mistypes and simplified the process of writing code efficiently. It also streamlined the process of debugging the application locally and was overall a great asset in developing the project.

I used GitHub[20] to version control the development of my project. This was an easy choice due to familiarity with the service and the ease to merge pull requests and commit code. I made use of branching to separate out developing features from the main live version of the site. However, as I developed this program on my own and integrity of the data is unimportant for the application, quite often I would commit directly to the master branch.

To host this website I used the PaaS (Platform as a service) web application service offered by Microsoft Azure[21]. Microsoft Azure already had a starting GitHub template to develop Flask web applications so it provided a great way to start development of my project and to ensure integration between the local and cloud environments. I could have chosen to use Azure's IaaS (Infrastructure as a service) virtual machines to set up the website. However, the scope and resource intensity of this project wasn't enough to justify the added time spent configuring the virtual machine. The PaaS approach had everything needed for the application and more. Azure's PaaS also integrated well with using GitHub. I was able to set the deployment option to continuously deploy from the main branch. Any changes to the application I made locally on PyCharm and committed to the master branch on GitHub would thus within minutes be live on the Internet. This provided instant feedback if there were discrepancies in dealing with the cloud environment compared to the local environment. With this all set up, I could focus solely on the development of my application without concerning myself with the underlying infrastructure.

Another service that I used from Azure was a Redis cache. This cache enabled a form of temporary data storage that was a middle ground between local variables in the code and permanent databases. Sometimes I would find that local variables did not store the information outside the application context as I required (different user instances of the website could not access data from other sessions), but a SQL database implementation was clunky and overkill for the simply data I needed storing. The Redis cache provided a great balance of cross session key-value storage that was permanent only to the specified time interval (the cache can specify how long the data is stored before being garbage collected). SQL databases were thus not needed for simpler getting and setting of key-value storage.

Another important technological component of the application were the large amount of custom domain names required. This was because I needed to simulate the separation of different first party and third party domains that were treated as individual entities by browsers. Browsers treat domains as distinct entities rather then the IP of the servers used due to the fundamentals of modern web design. Although IP addresses and domain names are many-to-many related, as most cloud computing is distributed between different servers, domain names are viewed as the distinction. For example, most companies have multiple different servers that due to the location and load of the network, serve the same requests. Google for instance responds to a request like `www.google.com` at any number of different servers. If could be as an example `356.123.567.198`, or `596.123.375.198`, or any number of other IP addresses depending on how saturated the servers are or where the user browsing Google is located. This distributed computing paradigm is required for the requests to be processed efficiently and within reasonable time to the user.

This knowledge can be used to the applications benefit. As I only have access to one IP address for the single server, I mapped multiple custom domains to the same IP address. Although the server processing the requests is the same, the different base domains are seen as separate web entities to the browser so they are considered as separate unrelated websites. All that is needed is to respond to requests different depending on the originating domain name (which I will detail later on). The different domain names thus seem to the browser for all intents and purposes to be separate unrelated websites.

To register domain names, I used the company Freenom[22]. They offer free domain names for certain top-level domains such as .tk. Therefore there was no cost to registering these domains which was important due to the large number of these domains that needed to be registered. All in all, I used around eighteen domain names.

Finally, I used Bootstrap[23] to create visually appealing HTML website layouts without having to program copious amounts of CSS. This improved the aesthetics of the application.

3.2.2 Third Party Libraries

In addition to this technology, I also used a variety of third party python libraries in order to develop my application.

Firstly, there was the question of SQL database storage. A database was required to store information on users who browse the web application. This was in order to create a history log of visited first party domains in a similar manner to a malicious third party tracking websites as described in the background. This information could then both be used for showing the recently visited sites to the user so they observe a visual representation of the tracking, as well as when benchmarking the users browser.

At first, I installed a python package called sqlite3[24]. This library allowed the access and modification of SQLite databases. This is a comparatively low level solution to storing data as you have to pass into python commands the SQL queries to execute. For example, what would intuitively be simple functions to view the recent history and to insert a visited site entry would be relatively complicated using this pure SQL query approach as shown below:

```
1  def add_visited_site(uuid, site):
2      conn = sqlite3.connect('database.sql')
3      c = conn.cursor()
4           c.execute('''INSERT INTO history VALUES  (?,?)''', (uuid,
            ↪  site))
5           conn.commit()
6      c.close()
7
8  def select_sites_by_user(uuid, count):
9          conn = sqlite3.connect('database.sql')
10         c = conn.cursor()
11         c.execute('''SELECT * FROM history WHERE uuid = ? ORDER BY
           ↪  ROWID DESC LIMIT ?''', (uuid, count))
12         site_list = c.fetchall()
13         conn.commit()
```

```
14          c.close()
15          return site_list
```

Eventually, I realized the impracticality and verboseness of this direct SQL querying and decided to use something higher level. Luckily there exists a python Package for Flask which similar to the models system used by Django, encapsulates SQL functionality in python functions. This is called Flask-SQLAlchemy[25]. Rather then dealing with the table creation and querying directly by SQL, Flask-SQLAlchemy allows you to write models that encapsulates this behavior. For instance, The Base History model used to store sites visited by users looks as follows:

```
class BaseHistory(db.Model):
    __abstract__ = True
    id = db.Column(db.Integer, primary_key=True)
    site = db.Column(db.String(URL_MAX_LENGTH))
    ip = db.Column(db.String(32))
    timestamp = db.Column(db.DATETIME, default=datetime.utcnow)
```

When Flask then creates the database object (referred to by db in the code), an SQL table with these parameters is initialized if not already present. Then, rather than having to use SQL queries to convolute the process of adding and viewing entries, I can simply use one line functions such as `History('www.visitedsite.com', '567.365.234.267', timestamp())` to create a new entry. Similarly selecting entries is equally as easy. A one line function call like `History.query.filter_by(ip='567.365.234.267')` is enough to retrieve the relevant entries (based on IP address in this example). This Flask-SQLAlchemy package thus simplifies database management considerably.

Another two python packages, WTForms[26] and Flask-WTF[27], were used to simplify the creation of forms for the user to submit data. This was used in the configuration page for the user to submit which tracking mode/technique the first party websites should be set to. These two packages in combination take the arduous process of validating forms and implements helpful methods to expedite the process. Therefore, similar to Flask-SQLAlchemy, a form structure can be specified as follows as a FlaskForm (for the configuration page):

```
1 from flask_wtf import FlaskForm
2 from wtforms import RadioField, IntegerField
3 from wtforms.validators import InputRequired, NumberRange
4
5 from settings import TRACKING_MODES
6
```

```
7
8  class TrackerForm(FlaskForm):
9      tracker = RadioField('Tracker Options',
10                          choices=[('1', TRACKING_MODES['1']),
11                                   ('2', TRACKING_MODES['2']),
12                                   ('3', TRACKING_MODES['3']),
13                                   ('4', TRACKING_MODES['4']),
14                                   ('5', TRACKING_MODES['5']),
15                                   ('6', TRACKING_MODES['6'])],
16                          default='1')
17     first_party_cookie_size = IntegerField('First Party Cookie Size',
       ↪  validators=[InputRequired('This field is required'),
       ↪  NumberRange(min=3, max=64, message='Must be a number between 3
       ↪  and 64')], default=12)
18     cookie_size = IntegerField('Third Party Single Cookie Size',
       ↪  validators=[InputRequired('This field is required'),
       ↪  NumberRange(min=2, max=64, message='Must be a number between 3
       ↪  and 64')], default=12)
19     split_cookie_size = IntegerField('Third Party Split/Chained Cookie
       ↪  Size', validators=[InputRequired('This field is required'),
       ↪  NumberRange(min=1, max=16, message='Must be a number between 1
       ↪  and 16')], default=3)
20     local_storage_super_cookie_size = IntegerField('Third Party Local
       ↪  Storage Super Cookie Size', validators=[InputRequired('This
       ↪  field is required'), NumberRange(min=2, max=64, message='Must
       ↪  be a number between 3 and 64')], default=12)
21     local_storage_split_super_cookie_size = IntegerField('Third Party
       ↪  Local Storage Split/Chained Cookie Size',
       ↪  validators=[InputRequired('This field is required'),
       ↪  NumberRange(min=1, max=16, message='Must be a number between 1
       ↪  and 16')], default=3)
```

On lines 17, 18, 19, 20 and 21, various WTForms validators, WTForms types and defaults can be set. For instance, the range of values that would be accepted by the user is restricted and the fields are set to be required. This also checks that the type of input typed in by the user into the form is correct (using the IntegerField). Having this form handling system simplified on the server side reduces the pain of coding various conditionals to handle invalid user submissions. Thus, all the validation is done via the custom error messages I typed and concerns about bugs from long user produced code validation, are eased.

All that is required, is to render this form in HTML by passing in the Form context to the required request handler. Then the visual layout of the form can be altered without the functionality being changed in the HTML template.

3.2.3 Design Overview

To ensure development of the project was iterative and easy to make alterations, I separated the web application into multiple different packages. On Flask these were registered as blueprints to be included in the main.py file. Each of these packages comprised a different section of the overall website. Flasks core components are the views which respond to requests via a URL pattern, and HTML templates. Therefore, each separate package handled requests from different domain names. For instance, `first_party` package handled requests from any of the first party base domains. `third_party` handled requests for single cookie tracking. `third_party_split` handled requests for split cookie tracking and so on and so forth.

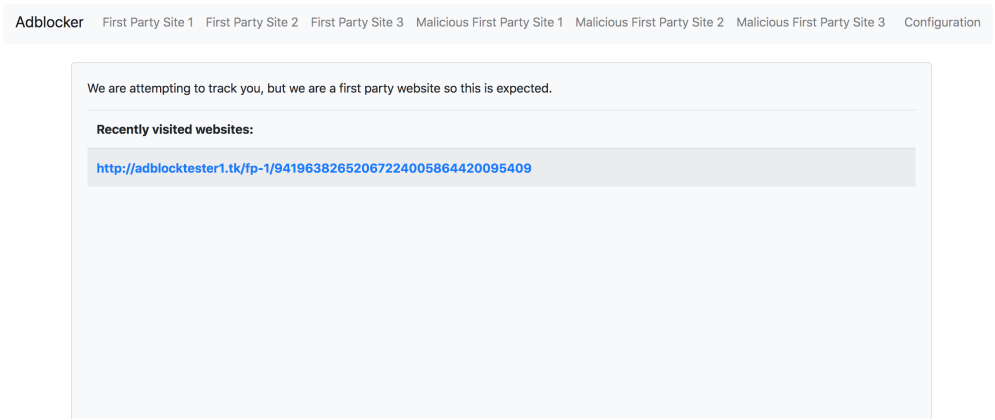
Each of the request handling methods in these views.py files accepted a matching URL pattern such as `/tp-single-cookie/<config_id>/`. All that was needed was to ensure that only the valid custom domain names for this particular package were accepted. In the case of `/tp-single-cookie/<config_id>/` for instance, we would only accept my third party single cookie domain which was `http://third-party-tracker-single-cookie.tk/`. Otherwise, we would return a 404 response to anyone who used the URL pattern with a different base domain. This ensured that the separate base domains seemed to operate on an entirely different web server as described earlier. This was done on the first line of every request handling method via a utility function shown below:

```
def check_base_domain(request, urls):
    if not request.url_root in urls:
        abort(404)
```

This method would check the URL request and ensure that it is the same as one of the given URL's. For instance, to ensure that the request for a single third party cookie was from the correct domain, the command I would use was `check_base_domain(request, 'http://third-party-tracker-single-cookie.tk/')`. To improve the code all these URL's to compare with the request were kept in the settings.py file as constants.

Another design consideration was to allow a user to test out the web application manually. In order to accomplish this, a navigation bar was to be present on first party domain templates as shown in figure 3.3. This navigation bar could link between two other first party domains and the configuration page (to set the tracking mode). Thus, all first party domains inherited from this HTML template, as did the configuration page:

Figure 3.3: The navigation bar present on all none third party domains



This design choice, ensured that anyone could test the current tracking mode (set in the configuration web page), by manually switching between the first party URL's and seeing if the third party domains were being blocked. This also ensured that it was easy to navigate around the web application as all the important links are constantly click-able.

To easily and cleanly program this into Flask's template structure, I used a base HTML file that implemented this navigation bar. All other HTML templates would thus inherit from this base file and add the other components that web page required.

Another design choice was how to separate the live and local versions of the website. Of course, in order to test the web application locally I had to use the local server. However, the web application relied on the interplay between several different domain names. The web application would therefore behave inconsistently if we only used the `127.0.0.1` address. To solve this issue, as in the prototype (section 3.1.2), I mapped domain names to the local server IP address using the `private/etc/hosts` file by adding the lines as shown below:

```
127.0.0.1       local.adblocktester1
127.0.0.1       local.adblocktester2
127.0.0.1       local.adblocktester3
127.0.0.1       local.third-party-tracker-single-cookie
127.0.0.1       local.third-party-tracker-super-cookie
127.0.0.1       local.third-party-tracker-split-1
127.0.0.1       local.third-party-tracker-split-2
127.0.0.1       local.third-party-tracker-split-3
127.0.0.1       local.third-party-tracker-split-4
127.0.0.1       local.third-party-tracker-split-super-1
127.0.0.1       local.third-party-tracker-split-super-2
```

```
127.0.0.1       local.third-party-tracker-split-super-3
127.0.0.1       local.third-party-tracker-split-super-4
127.0.0.1       local.third-party-tracker-master-join
127.0.0.1       local.adblocktesterconfig
127.0.0.1       local.adblocktestermalicious1
127.0.0.1       local.adblocktestermalicious2
127.0.0.1       local.adblocktestermalicious3
```

I made these local domain names as similar as possible to the custom domain names for the live website. Now all that was needed was to use either use these mapped local domains, or the custom live website domains depending on whether the website was on my machine or the live site respectively. This was done via an environment variable. Therefore, as shown in a fragment of the settings.py file, the domains are set to the correct values:

```
DEBUG = os.environ['APPLICATION_CONFIGURATION'] ==
↪  'app_config.DevConfig'

URLS = {
    'FP_URL_1': 'http://adblocktester1.tk/' if DEBUG is False else
    ↪  'http://local.adblocktester1:5000/',
    'FP_URL_2': 'http://adblocktester2.tk/' if DEBUG is False else
    ↪  'http://local.adblocktester2:5000/',
    'FP_URL_3': 'http://adblocktester3.tk/' if DEBUG is False else
    ↪  'http://local.adblocktester3:5000/',

    'FP_URL_MALICIOUS_1': 'http://adblocktestermalicious1.tk/' if
    ↪  DEBUG is False
    else 'http://local.adblocktestermalicious1:5000/',
    'FP_URL_MALICIOUS_2': 'http://adblocktestermalicious2.tk/' if
    ↪  DEBUG is False
    else 'http://local.adblocktestermalicious2:5000/',
    'FP_URL_MALICIOUS_3': 'http://adblocktestermalicious3.tk/' if
    ↪  DEBUG is False
    else 'http://local.adblocktestermalicious3:5000/',

    'TP_URL': 'http://third-party-tracker-single-cookie.tk/' if DEBUG
    ↪  is False
    else 'http://local.third-party-tracker-single-cookie:5000/',

    'TP_SPLIT_URL_1': 'http://third-party-tracker-split-1.tk/' if
    ↪  DEBUG is False
    else 'http://local.third-party-tracker-split-1:5000/',
```

```
'TP_SPLIT_URL_2': 'http://third-party-tracker-split-2.tk/' if
↪  DEBUG is False
else 'http://local.third-party-tracker-split-2:5000/',
```

Note that this is only a subsection of the URLS dictionary that is used map the custom domains. Nonetheless, this shows the principle of setting the URL's depending on the environment variable which sets DEBUG to True if on the local machine, and False if on the live website).

Chapter 4

Implementation

In this section I will detail the implementation of the configuration page, the seven different tracking modes, and finally the benchmark.

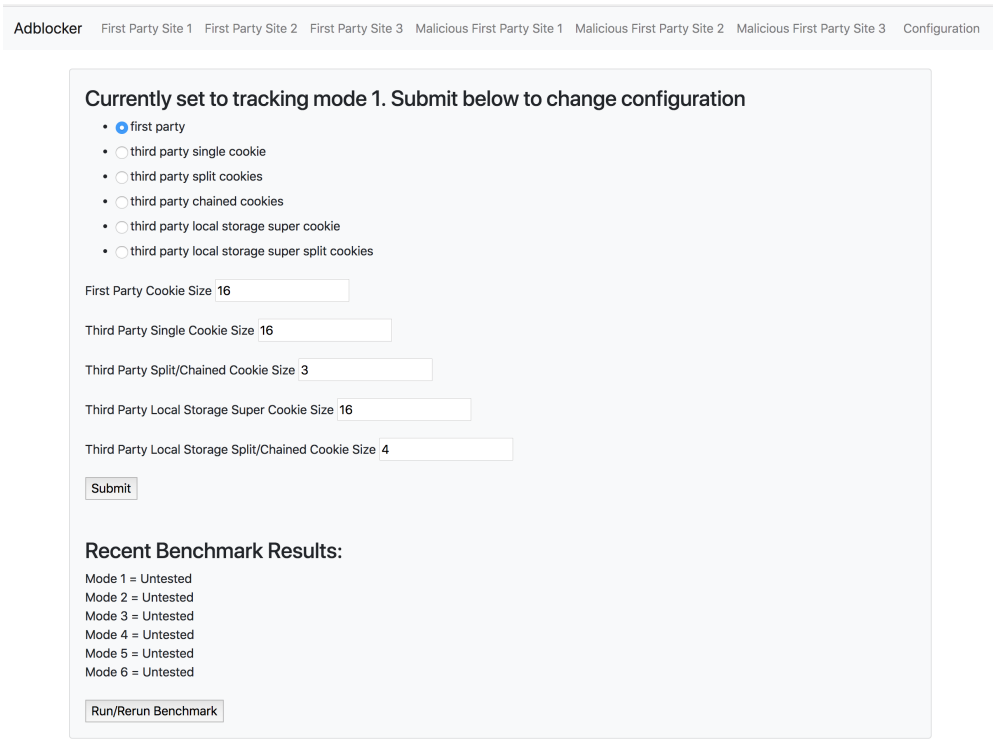
4.1 Configuration Page

This page is handled by the config package which is registered as a Flask blueprint. This page/package allows the user to specify the configuration of the web application. There are many different aspects to this package including forms, models (SQLAlchemy databases) and views.

To start with the user visits this page at `http://adblocktesterconfig.tk`. This page will then set a first party cookie (adblockers should allow cookies on first party sites). This cookie is a 64 length integer and uniquely identifies the user. This identification is needed to ensure that each user can customize their web application experience without interfering with concurrent users test environments. This identifier is also passed as a parameter to all the other links on the navigation bar so that the first party sites know how to act based on the users preferences. This cookie creates (if first time visitor) or retrieves a user database entry which sets the relevant fields. These fields include the selected tracking mode, the cookie size for each of the tracking modes, as well as the result of the benchmark for each of the modes.

On this page, as shown in figure 4.1, users have the option to change the currently selected third party tracking mode from the six options. In addition, users can change the length of each modes cookies, this allows more specification to test the limits of the adblockers and what cookies sizes they will accept as none tracking. Once this form is submitted, these changes are applied to the user database entry. Now, when the first party sites are clicked on, it will retrieve these newly set preferences from the user database entry which determines correct tracking mode to implement.

Figure 4.1: The configuration page



As well as being able to submit these changes to the configuration which alters the database, users can also run or rerun the benchmark test (as described later in this chapter) which cycles through the modes of tracking and returns whether the tracking mode was a success (the users current adblocker/browser was unable to stop the tracking) or a failure (the users current adblocker/browser was able to stop the tracking and ensure privacy). Also, the way I implemented the benchmark is that it uses the current configuration cookie size settings while cycling through the modes. This enables further customization of the benchmark to tailor the benchmarking to the required cookie entropy.

4.2 First Party Only (Mode 1)

This type of tracking sets only first party cookies. There are no iframes to external sites when set to this mode so the only privacy information surrendered is to the first party site itself. This is expected behavior of browsing a first party sites, we assume that they can create a history log of all the visited sites under its control. This mode should thus when benchmarked return ”Success” unless all cookies are blocked, even those which are not third party. By not allowing first party cookies, key functionality of websites might not work so this is not an ideal configuration. Although this guarantees a high level of privacy, it also sacrifices on usability.

4.3 Third Party Single Cookie (Mode 2)

This mode is the traditional technique that third party websites will use to track users. The first party websites I implemented when set to this mode will iframe in a third party domain as follows:

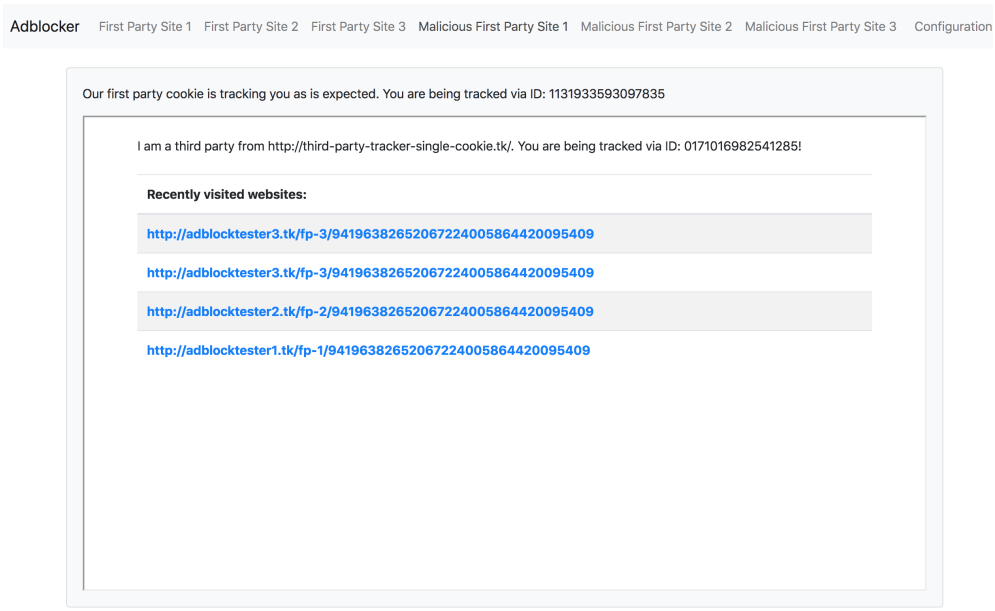
```
<iframe src="{{ urls.TP_URL }}tp-single-cookie/{{ config_id }}"
style="position:relative; width: 100%; height: 600px;">
</iframe>
```

Then, this iframe causes the third party single cookie URL to respond and set a cookie. The Flask code that the third party website responds to is shown below:

```
1  @tp.route('/tp-single-cookie/<config_id>')
2  def tp_single_cookie(config_id):
3      check_referer(request, URLS['TP_URL'])
4      config_cookie_length = implicit_user_login(User,
       ↪ config_id).cookie_size
5      uuid = request.cookies.get('id')
6      trackable_user = implicit_user_login(TrackableUUID, uuid,
       ↪ config_cookie_length)
7
8      log_site_visit(History, request.referrer, trackable_user,
       ↪ request.remote_addr)
9      recent_history = generate_recent_history(History, trackable_user,
       ↪ 10)
10     if uuid is not None:
11         redis_set_benchmark_recent_site(config_id, '2',
           ↪ request.referrer)
12
13     response = make_response(
14         render_template(THIRD_PARTY_SINGLE_TEMPLATE,
           ↪ history_log=recent_history, current_url=request.url_root,
15                         cookie_id=uuid))
16     if uuid is None:
17         response = append_cookie(response, trackable_user.uuid)
18     return response
```

The most important lines of code are 5 and 6 which gets the cookies value (unique identifier) and finds the user from the database with this corresponding value. Then, a history log entry is registered via the `log_site_visit` function on line 8. As explained in the Background section, we can gather information on the first party site visited by using the HTTP referer header tag. This is done on Flask by accessing the header field via the command `request.referer` (also line 8).

Figure 4.2: Mode 2. This mode displays the history log



Then on line 9, we generate a recent history log for this user, this can then be rendered on the HTML template to detail the information that this third party tracker was able to gather from this type of tracking.

Finally, on line 16-18, we set the cookie if this is a new user for this third party website.

4.4 Third Party Split Cookies (Mode 3)

In this mode, I used the tracking technique from the initial prototype as mentioned earlier, but fleshed out on the server side to generate a history log. To start with, instead of iframing in a single third party domain with one single cookie, the first party domain iframes in four separate third party domains as shown below:

```
<div class="modal-body row">
<div class="col-md-6">
<iframe src="{{ urls.TP_SPLIT_URL_1 }}tp-split-1/{{ config_id }}"
style="position:relative; width: 100%; height: 300px;">
</iframe>
<iframe src="{{ urls.TP_SPLIT_URL_3 }}tp-split-3/{{ config_id }}"
style="position:relative; width: 100%; height: 300px;">
</iframe>
</div>
<div class="col-md-6">
```

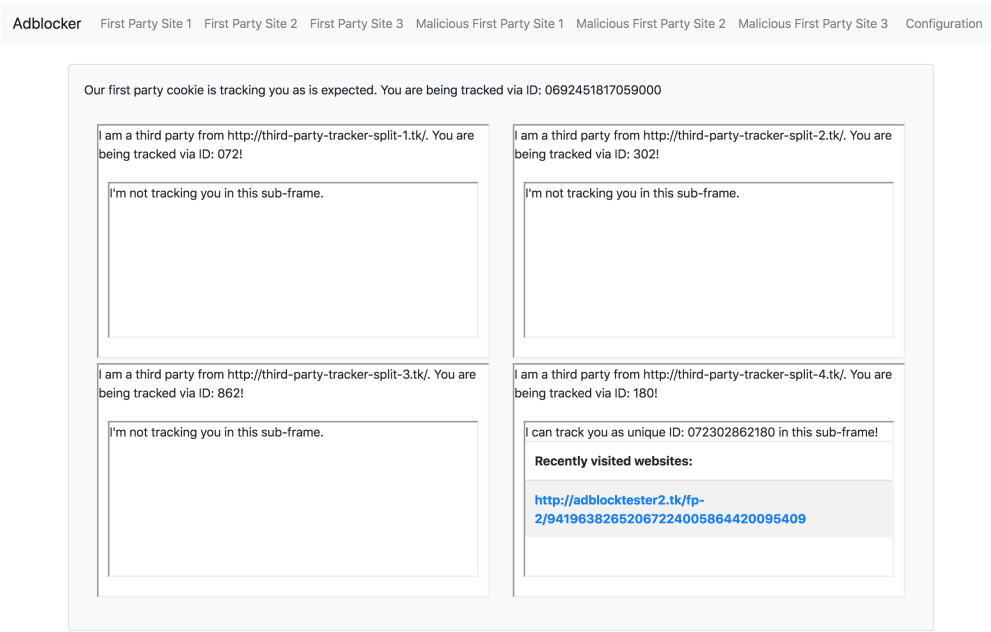
```
<iframe src="{{ urls.TP_SPLIT_URL_2 }}tp-split-2/{{ config_id }}"
style="position:relative; width: 100%; height: 300px;">
</iframe>
<iframe src="{{ urls.TP_SPLIT_URL_4 }}tp-split-4/{{ config_id }}"
style="position:relative; width: 100%; height: 300px;">
</iframe>
</div>
</div>
```

Each of these four third party domains are handled separately by different domains but function in a similar manner. The fourth of the four third party domains is handled as shown below:

```
@tps.route('/tp-split-4/<config_id>')
def tp_split_4(config_id):
    check_referer(request, URLS['TP_SPLIT_URL_4'])
    config_cookie_length = implicit_user_login(User,
    ↪ config_id).split_cookie_size

    uuid = get_uuid_from_cookies(request, 'id', config_cookie_length)
    trackable_uuid_1 = SplitTrackableUUID1.get_or_create(uuid,
    ↪ config_cookie_length)

    log_site_visit(SplitHistoryBase1, request.referrer,
    ↪ trackable_uuid_1, request.remote_addr)
    redis_register_split('4', request.remote_addr,
    ↪ trackable_uuid_1.uuid, request.referrer)

    safe_referer = quote(request.referrer)

    response = make_response(
        render_template(THIRD_PARTY_SPLIT_TEMPLATE,
        ↪ current_url=request.url_root, urls=URLS, cookie_id=uuid,
                        safe_referer=safe_referer, index=4,
                        ↪ config_id=config_id))
    if uuid is None:
        response = append_cookies(response, 'id',
        ↪ trackable_uuid_1.uuid)
    return response
```

This functions very similarly to Mode 2 with a smaller cookie size (and four separate domain instances). However, a key difference is the redis cache which is used to register the cookie ID used for this IP address and first party domain visit.

Figure 4.3: Mode 3. This mode displays the history log within the master third party domain which is iframed within the split third party domains



Therefore, I used these four domains to call a master which pieces together the cookie; we thus simply iframe within the four separate third party websites a different master third party website as shown below:

```
<iframe src="{{ urls.TP_MASTER_URL }}tp-master/{{ config_id }}/{{
 ↪  index }}/safe_referer?id={{ safe_referer }}" style="position:
 ↪  relative;
height: 200px; width: 95%; top: 0%; right: 2.5%; left: 2.5%;">
</iframe>
```

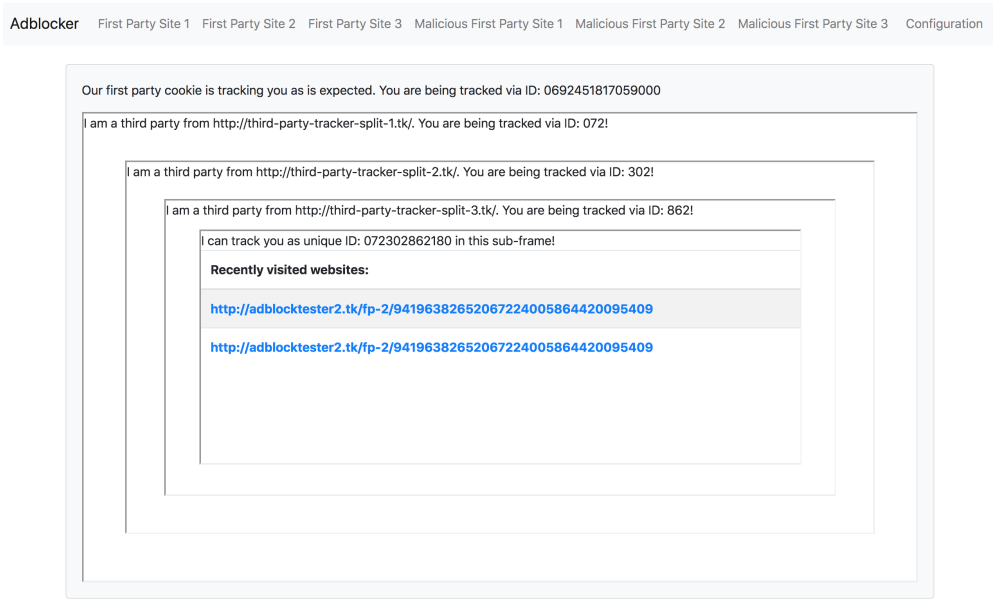
This master third party domain thus processes this request from the original third party domain and uses the redis cache to string together the four separate portions of the cookie id to create a master unique identifier. This is then used to log the history of the site visited.

4.5 Third Party Chained Cookies (Mode 4)

This works extremely similarly to Mode 3, except instead of four separate iframes that all call a master to stitch the cookie value together (the unique identifier), a chain of iframes within iframes is made to build up the cookie value.

To start off, the initial third party iframe is loaded and sets a low entropy cookie. This then loads the second third party iframe:

Figure 4.4: Mode 4. This mode visualizes the recursive iframe within iframe approach of building the combined unique identifier value to sufficient length in low entropy cookie chunks




```
<iframe src="{{ urls.TP_SPLIT_URL_2 }}tp-split-chain-2/{{ config_id
 ↪  }}/{{ combined_id }}/safe_referer?ref={{ safe_referer }}"
style="position: absolute; height: 80%; width: 90%; top: 10%; left:
 ↪  5%;">
</iframe>
```

The combined ID variable that is passed to the URL as shown above, is the cookie ID. Then this second third party website iframe that was loaded uses the combined ID and appends its own low entropy cookie. This is then passed on in the exact same manner to the 3rd third party iframe:

```
<iframe src="{{ urls.TP_SPLIT_URL_3 }}tp-split-chain-3/{{ config_id
 ↪  }}/{{ combined_id }}/safe_referer?ref={{ safe_referer }}"
style="position: absolute; height: 80%; width: 90%; top: 10%; left:
 ↪  5%;">
</iframe>
```

Now the combined ID variable contains the concatenation of the first iframes cookie value and the second iframes cookie value. We thus repeat this recursive process until the cookie is of sufficient length as shown in figure 4.4. Once the cookie is big enough we create a history log in the same manner as the other tracking techniques.

4.6 Local Storage Super Cookie (Mode 5)

Instead of traditional cookies, this mode relies on HTML5 being able to store key value information per domain (iframe). As localStorage is client side, I used JavaScript to generate the third party domains unique user identifier, and then redirected this to a different request:

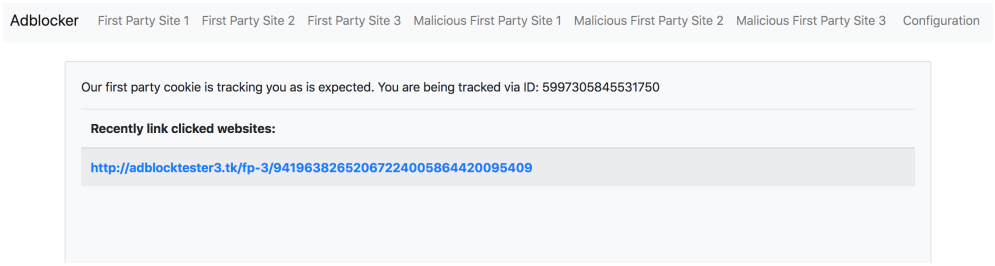
```
1  window.onload = function () {
2  test = localStorage.getItem("u");
3  if (test === null) {
4  var possible = "0123456789";
5  var text = "";
6  for (var i = 0; i < {{ config_cookie_length }}; i++)
7  text += possible.charAt(Math.floor(Math.random() * possible.length));
8  localStorage.setItem("u", text);
9  }
10 var next_site = '{{ next_site }}/' + localStorage.getItem("u") +
   ↪  '/safe-referer?id=' + '{{ safe_referer }}';
11 setTimeout(function () {
12 window.location = next_site;
13 }, 1000);
14 };
```

The for loop in lines 6-7 creates the random numeric unique identifier. then line 8 sets this value to key ”u”. Then, lines 10-14 redirect to the next site. This redirect parameterizes the client localStorage user identifier inside the URL so the function handling this redirect has access to the unique user identifier and can respond by creating a history log accordingly. Therefore, this information has been transferred from the client to the server side via a redirect. Of course, alternative methods other than redirects can be used - for example an AJAX function. All that matters is now this client side unique identifier has been transferred to the server with a simple redirect, and thus a history log can be computed as in all the previous tracking modes.

4.7 Local Storage Chain Cookies (Mode 6)

This tracking technique is an amalgamation of Modes 4 and 5. A smaller low entropy unique identifier is used by using the client side localStorage redirect approach as in mode 5. And then after the redirect, another iframe is loaded to repeat the same Mode 5 process to build up the combined ID (in a recursive manner equivalent to mode 4). After the Combined ID is big enough to be a unique identifier the recursion is stopped and a history log can be made.

Figure 4.5: Link Shimming. This malicious first party website has recorded the external links clicked on as shown below



4.8 Link Shimming First Party

This is the seventh and final tracking technique implemented. In order to properly implement first party link shimming I had to modify the navigation bar for these malicious first party domains. Now instead of each first party link having a href only, the link also contains a data-lynx-uri field. This is shown below:

```
<a id="fp-link-1" class="nav-link" href="{{ url_malicious }}{{
 ↪  config_id }}/u?id={{ url_fp1 | quote }}"
data-lynx-uri="{{ url_fp1 }}">
First Party Site 1</a>
```

The href now contains the malicious link shimming URL which will log the external link, whereas the intended link is stored in the data-lynx-uri. Now all that is required is to trick the user by displaying the normal looking link when the mouse is hovering over it. This is done via JavaScript:

```
function replaceURI(link_id) {
var new_uri = document.getElementById(link_id).href;
var new_href =
 ↪  document.getElementById(link_id).getAttribute("data-lynx-uri");
document.getElementById(link_id).setAttribute("data-lynx-uri",
 ↪  new_uri);
document.getElementById(link_id).setAttribute("href", new_href);
return false;
}
```

This simple script replaces the data-lynx-uri with the href. Therefore, when hovering over the link, it looks like the external link. But when clicked on, we use this function again to swap URLs and deceptively follow the link shimming malicious URL which first registers the external link we clicked on to a history log, before redirecting to the intended link. The malicious first party domain now knows which external links you have clicked on and can display this to you as in figure 4.5 above.

Figure 4.6: Benchmarking. This currently running benchmark shows Mode 1 was successful in tracking the user, whereas the other techniques are still in progress

4.9 Benchmarking

To run a benchmark you simply click the Run/ReRun Benchmark button from the configuration page. This benchmark works rather simply by setting a nested redirect loop. The outer loop sequentially goes through each of the tracking modes testing them and producing a ”Success” or ”Failure” state, and the inner loop goes through a cycle of numerous first party sites to detect if the history log is up to date (which ensures the third party was able to track you and thus ”Success”). These first party sites are rendered on 1 by 1 pixels to appear invisible to the user, and instead progress text which details the successes and failures up to that point is displayed.

This essentially automates the process of visiting the first party sites yourself via the navigation bar, and checking to see if the history log was being updated or whether the third party iframes were being blocked. The redis cache is used to keep the most recently visited sites in memory for the benchmark to assess.

The most recently run benchmark is saved in the configuration page so you can always know the current browser configurations ability to block these tracking techniques.

Initially, I intended to make the malicious web application distinct from the benchmark. The benchmark would be run locally as a software application that utilized a web driver. This web driver would specify the adblocker to test and then do the redirect loop as discussed above. However, this would complicate the process of benchmarking. Not only would it be a pain to express extensions in the local application, it wasn't intuitive. A user of my web application would rather not go through the effort of having to download software to his/her computer before being able to benchmark their specific adblockers. Also, making the application software on the computer meant that I would need different versions for different operating systems.

With this reasoning, I made the benchmark a part of the web application which significantly streamlined and improved the benchmarking process.

Chapter 5

Adblocker Analysis

5.1 Benchmarking Privacy Badger

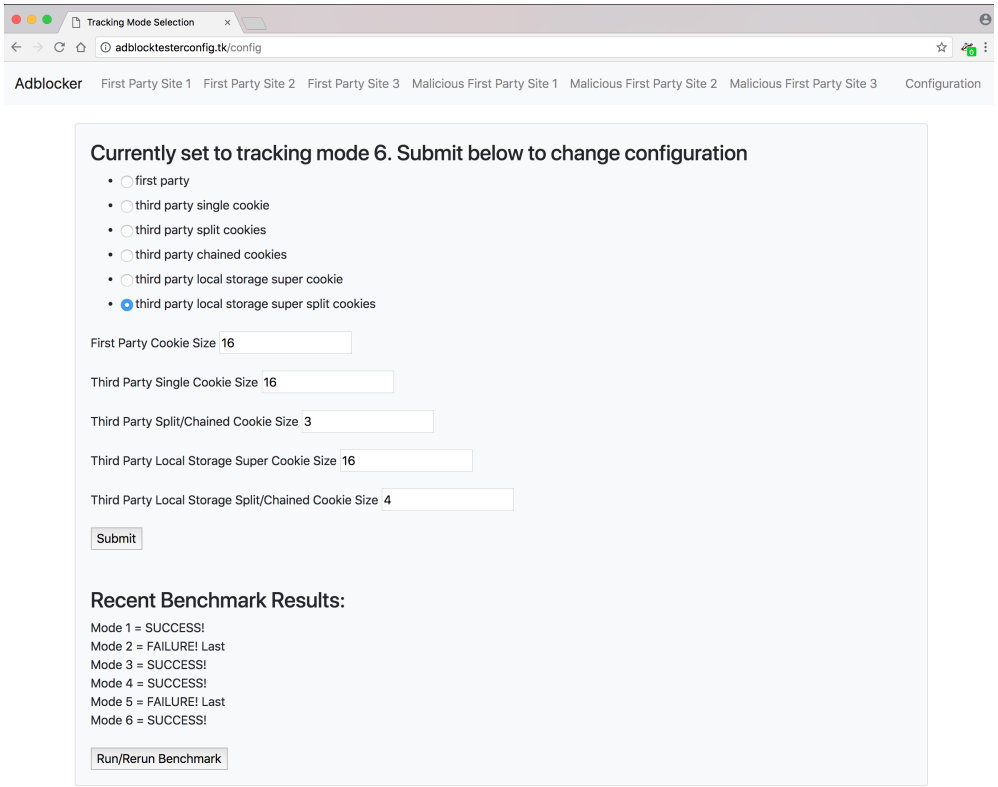
When running the benchmark with privacy badger installed on google chrome, privacy badger prevented modes 2 and 5. Therefore, normal single cookie tracking, and super cookie HTML5 localStorage tracking were prevented. This is due to the nodal analysis Privacy Badger performs on third party domains to prevent cross-site tracking. However, modes 3, 4 and 6 circumvented Privacy Badger's low entropy filter (the cookie deletion heuristic) and were all able to track the user. Therefore, splitting the unique identifier into separate chunks which seemed to come from unrelated domains worked in practice. To remedy this Privacy Badger should sum up all the third party cookie entropies and if a certain threshold is reached block all of them. This unfortunately would mean if multiple third party cookies are set, even if they are none tracking, they would be blocked. Thus third party cookies cannot be allowed if we needed to guarantee privacy. Most mobile phones already do block all cross-site third party cookies as in Safari's mobile browser but as I have shown in Mode 5 and 6, tracking can still be accomplished without traditional cookies.

Also, link shimming was undetected by Privacy Badger. Although for Facebook and Twitter, Privacy Badger had custom made methods to replace the href with the true external link - this custom method only applies to those websites and not generic link shimming.

5.2 Benchmarking Other Adblockers

The other adblock extensions I tested were Adblock and uBlock and they do not stop any of these tracking techniques whatsoever. These adblockers only operate on a blacklist and make no attempt to use algorithmic methods to detect tracking. Therefore privacy is not of the highest priority for these popular adblockers.

Figure 5.1: Privacy Badger's Benchmark results.



Chapter 6

Evaluation

Overall the core strength of my web application was that it was able to quantify various tracking techniques. For instance, I was able to run the benchmark to assess the strengths and weaknesses of Privacy Badger, and in particular the prominent weaknesses in traditional adblockers like Adblock and uBlock. This form of analysis, whereby the techniques are being quantified rather than the contextual application in the real world - provides a methodological distinction compared to other privacy research [1].

Of course, there are areas of my work in which there could be improvements. Not all the techniques I wanted to implement made it in time. In particular other super cookie methods, such as Flash and Silverlight were not present. Also, I was unable to exploit or test fingerprinting, which forms a core component Privacy Badgers code. If some sort of fingerprint testing was present on the web application, I would have covered nearly all ways Privacy Badger attempts to detect tracking.

In terms of functionality of the website there are also minor annoyances in the end product. This is mainly caused by issues with the speed and responsiveness of the application. As I decided on PaaS rather than IaaS and I didn't use a very resource capable server, the website can take a while to respond. Especially if the website hasn't been requested for quite some time. Even more annoyingly the location of the servers are in Central US rather than UK South. This further impacted performance negatively. That being said most responses were within a couple seconds. In addition, a rather annoying development discovery, was that Imperial College blocks .tk domains. This means that I have to user my home IP address to test the application (or do it locally on my laptop). The alternative would be to use top level domains like .com, however, the cost of the numerous domain names would be too great.

The main competitor I found on-line which similarly benchmarks the browser against tracking techniques is the Panopticlick[28]. As I was inspired by the redirect method Panopticlick uses, it is similar in some respects to my program, however only two

tracking modes are tested. Setting a single long cookie (mode 2) and fingerprinting. The only other test is checking whether the website respects the DNT policy (Do Not Track) which isn't a true tracking technique. Also, all the logic is done client side on Panopticlick which means it isn't as feature rich as my application. However, this does mean that Panopticlick will not run into data storage issues and can also process requests quicker than my application. Also, my application can test browsers manually by the user and is much more customizable (via the configuration page). Also, malicious first party link shimming is implemented as well as ways to exploit Privacy Badger whereas Panopticlick only considers the well established third party tracking techniques (other than fingerprinting).

Chapter 7

Conclusion

In conclusion, I was able to create a set of malicious tracking websites. Using the inbuilt benchmark these tracking techniques could be tested on a browser setup to evaluate any algorithmic blocking the browser or adblock extensions attempted. Testing Privacy badger with this benchmark, I found weaknesses and strengths in its design; in particular the heuristic to detect cookies (and localStorage) I found to be flawed. All things being considered, this web application functions well and has a nice design. It is also intuitive to use. However, some features are missing which I would have liked to have added.

If I were to do this project again (or had more time), I would have liked to add the fingerprinting tracking technique. Although quite complicated, this is a really exciting way to surreptitiously track a user via a HTML exploit. Also, other super cookie modes such as flash and silverlight could have also been added to further add depth to the quantity and scope of this web application.

However, I am thinking of open sourcing the project which should mean due to the iterative nature of adding tracking techniques, any missed opportunities can be added. This project even has potential in the future when new and exciting methods are discovered that could track users. With this proper open source implementation, this should help developers designing privacy based adblockers to improve their software.

Chapter 8

User Guide

To start with, clear your browser of any cookies and install the extensions you want to test. Then navigate to the configuration page at `http://adblocktesterconfig.tk`. Firstly, you should setup the cookie sizes to the values you want (keep in mind cookie size is the length of the uuid string). Then submit to confirm these changes. There are now two options on how to use the web application. Either set the tracking mode and then submit for a manual experience, or simply instead run the automatic benchmark.

If you want to manually use the site, set the tracking mode to the parameters you require and submit.
Now you can observe the results by navigating between the first party websites on the navigation bar at the top of the screen. This simulates the first party websites under the tracking mode you chose. Keep in mind the malicious first party sites won't be affected by the tracking mode, these implement an entirely separate tracking paradigm called link shimming.
You should be able to see as you browse these first party sites history logs build up underneath with text detailing the domain that is doing the tracking.
At any point you can navigate to the configuration page at the top right of the screen, or to any of the other first party sites.

If you want to benchmark, go to the configuration page and simply click the Run/ReRun Benchmark button. Do not change the URL address and let the web application progress through the benchmark. It should take roughly a minute to benchmark. When finished, it will redirect you back to the configuration page. You should now be able to see the results of the benchmark next to the related mode. "Success", means the tracking technique successfully tracked you. "Failure", means the tracking technique failed to track you (your adblocker worked to prevent this tracking technique).

Bibliography

[1] Gervais Arthur, Filios Alexandros, Lenders Vincent, and Srdjan Capkun. Quantifying web adblocker privacy, 2016. URL https://eprint.iacr.org/2016/900.pdf. Accessed on 14.06.2018. pages i, 10, 42

[2] Privacy Badger. Privacy badger blocks spying ads and invisible trackers, 2018. URL https://www.eff.org/privacybadger. Accessed on 14.06.2018. pages 1, 2, 11

[3] Adblock. Adblock. block ads. browse faster., 2018. URL https://getadblock.com. Accessed on 14.06.2018. pages 2

[4] uBlock. Block ads and trackers. browse faster., 2018. URL https://www.ublock.org. Accessed on 14.06.2018. pages 2

[5] Electronic Frontier Foundation. About eff, 2018. URL https://www.eff.org/about. Accessed on 14.06.2018. pages 2, 11

[6] Alexa. The top 500 sites on the web, 2018. URL https://www.alexa.com/topsites. Accessed on 14.06.2018. pages 2

[7] SeleniumHQ. What is selenium?, 2018. URL https://www.seleniumhq.org. Accessed on 14.06.2018. pages 3

[8] Google. Digital marketing solutions, 2018. URL https://www.doubleclickbygoogle.com/solutions/digital-marketing/. Accessed on 14.06.2018. pages 5

[9] Gibbs Samuel. Mobile web browsing overtakes desktop for the first time, 2016. URL https://www.theguardian.com/technology/2016/nov/02/mobile-web-browsing-desktop-smartphones-tablets. Accessed on 15.06.2018. pages 8

[10] w3schools. Html5 web storage, 2018. URL https://www.w3schools.com/html/html5_webstorage.asp. Accessed on 15.06.2018. pages 8

[11] Acar1 Gunes, Eubank Christian, Englehardt Steven, Juarez Marc, Narayanan Arvind, and Diaz1 Claudia. The web never forgets: Persistent tracking mechanisms in the wild, 2014. URL https://securehomes.esat.kuleuven.be/~gacar/persistent/the_web_never_forgets.pdf. Accessed on 15.06.2018. pages 8

[12] Bennett Cyphers. Privacy badger rolls out new ways to fight facebook tracking, 2018. URL https://www.eff.org/deeplinks/2018/05/privacy-badger-rolls-out-new-ways-fight-facebook-tracking. Accessed on 15.06.2018. pages 8

[13] Matt Jones. Link shim - protecting the people who use facebook from malicious urls, 2012. URL https://www.facebook.com/notes/facebook-security/link-shim-protecting-the-people-who-use-facebook-from-malicious-urls/10150492832835766/. Accessed on 15.06.2018. pages 9

[14] Rhana Cassidy. A how to guide to ad blocking, 2017. URL https://blog.getadblock.com/a-how-to-guide-to-ad-blocking-be452ed5ed6f. Accessed on 14.06.2018. pages 9

[15] EasyList. Overview, 2018. URL https://easylist.to. Accessed on 14.06.2018. pages 9

[16] Privacy Badger. Cookie test cases, 2015. URL https://github.com/EFForg/privacybadger/pull/705. Accessed on 16.06.2018. pages 17

[17] Flask. Flask is a microframework for python based on werkzeug, jinja 2 and good intentions. and before you ask: It's bsd licensed!, 2018. URL http://flask.pocoo.org. Accessed on 17.06.2018. pages 20

[18] Django. The web framework for perfectionists with deadlines., 2018. URL https://www.djangoproject.com. Accessed on 17.06.2018. pages 20

[19] JetBrains. Pycharm. python ide for professional developers, 2018. URL https://www.jetbrains.com/pycharm/. Accessed on 17.06.2018. pages 20

[20] GitHub. Github brings together the world's largest community of developers to discover, share, and build better software, 2018. URL https://github.com. Accessed on 17.06.2018. pages 20

[21] Microsoft Azure. Your vision. your cloud., 2018. URL https://azure.microsoft.com/en-gb/. Accessed on 17.06.2018. pages 20

[22] Freenom. Freenom - a name for everyone, 2018. URL http://www.freenom.com/en/index.html?lang=en. Accessed on 17.06.2018. pages 21

[23] Bootstrap Core Team. Build responsive, mobile-first projects on the web with the world's most popular front-end component library., 2018. URL https://getbootstrap.com. Accessed on 17.06.2018. pages 22

[24] Gerhard Hring. sqlite3 db-api 2.0 interface for sqlite databases, 2018. URL https://docs.python.org/2/library/sqlite3.html. Accessed on 17.06.2018. pages 22

[25] Flask-SQLAlchemy. Flask-sqlalchemy is an extension for flask that adds support for sqlalchemy to your application, 2018. URL http://flask-sqlalchemy.pocoo.org/2.3/. Accessed on 17.06.2018. pages 23

[26] WTForms Team. Wtforms documentation. this is the documentation for wtforms 2.2.1., 2010. URL `https://wtforms.readthedocs.io/en/stable/`. Accessed on 17.06.2018. pages 23

[27] DuPlain Dan, Jacoband Ron, Lepage Daniel, Ford Anthony, Yang Hsiaoming, and Lord David. Simple integration of flask and wtforms, including csrf, file upload, and recaptcha., 2018. URL `http://flask-wtf.readthedocs.io/en/stable/`. Accessed on 17.06.2018. pages 23

[28] ELECTRONIC FRONTIER FOUNDATION. Is your browser safe against tracking?, 2018. URL `https://panopticlick.eff.org`. Accessed on 17.06.2018. pages 42