UNIVERSITY OF CALIFORNIA SAN DIEGO

Names to Conjure With: Measuring and Combating Online Adversaries by Examining Their Use of Naming Systems

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy

in

Computer Science

by

Audrey Randall

Committee in charge:

Professor Aaron Schulman, Chair
Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair
Professor Kimberly Claffy
Professor Margaret E. Roberts

2023

The Dissertation of Audrey Randall is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

This dissertation is dedicated to the San Diego Bisikleta Club, the UCSD Race Condition running club, and the unofficial CSE swim crew. You never failed to support, challenge, and rejuvenate me during difficult times, and without you, both my PhD and my races would have been completed much more slowly! May the wind be at your backs, the sun be out of your eyes, your recoveries swift and your injuries few; and in particular, may you never lose the joy we share in the sports we love.

EPIGRAPH

"Words are pale shadows of forgotten names. As names have power, words have power. Words can light fires in the minds of men. Words can wring tears from the hardest hearts. There are seven words that will make a person love you. There are ten words that will break a strong man's will. But a word is nothing but a painting of a fire. A name is the fire itself."

– Patrick Rothfuss, *The Name of the Wind*

"All things are defined by names. Change the name, and you change the thing."

– Sir Terry Pratchett, *Pyramids*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to acknowledge my advisers, Aaron Schulman, Stefan Savage, and Geoffrey M. Voelker. Having no fewer than three advisers is a graduate school superpower, and having three such excellent advisers is a stroke of unimaginable good fortune that I certainly did not deserve. Quite apart from their technical prowess, research chops, and unfailing support, the amount of laughter they brought to the last five years cannot be overvalued. Thank you for the 4 A.M. nights and the pun wars in group meetings, the department hikes and the holiday party skits, and for everything else that made going to grad school the best decision I ever made.

Second, my undying gratitude goes to Robert Ariniello, who went through this process before I did and kept me cheerful and sane along the way. I also want to thank my family, who endured long-winded and excited explanations in excruciating technical detail for five years, and never tired of attempting to understand them.

Third, I want to thank all of my co-authors: Enze "Alex" Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Alisha Ukani, Peter Snyder, Alex C. Snoeren, and Wes Hardaker, and of course also Geoff, Stefan, and Aaron. This work would have been impossible without you, especially as I can't claim to have come up with the breakthroughs that made my first two papers possible on my own. Credit for those goes to Rama and Aaron, respectively.

Finally, I wish to thank all of the students, coworkers, and friends who have helped me along my journey — whether through giving advice, racing me on bikes, sharing their snacks, belaying me on very tall cliffs, or lending an ear when I needed one. To all the occupants of office 3140, and also or especially to Ben Du, Amanda Tomlinson, Alisha Ukani, Ariana Mirian, Chris Ye, and Zac Blanco, my everlasting gratitude for your help and support.

VITA

2018 Bachelor of Science, University of Colorado Boulder

2021 Master of Science, University of California San Diego

2023 Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Audrey Randall, Wes Hardaker, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. 2022. *The Challenges of Blockchain-Based Naming Systems for Malware Defenders.* In Proceedings of the APWG eCrime Conference (eCrime '22).

Audrey Randall, Peter Snyder, Alisha Ukani, Alex C. Snoeren, Geoffrey M. Voelker, Stefan Savage, and Aaron Schulman. 2022. *Measuring UID Smuggling in the Wild.* In Proceedings of the ACM Internet Measurement Conference (IMC '22).

Audrey Randall, Enze Liu, Ramakrishna Padmanabhan, Gautam Akiwate, Geoffrey M. Voelker, Stefan Savage, and Aaron Schulman. 2021. *Home is Where the Hijacking is: Understanding DNS Interception by Residential Routers.* In Proceedings of the ACM Internet Measurement Conference (IMC '21).

Audrey Randall, Enze Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M. Voelker, Stefan Savage, and Aaron Schulman. 2020. *Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers.* In Proceedings of the ACM Internet Measurement Conference (IMC '20).

Louis F. DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K. Saul, Aaron Schulman, Geoffrey M. Voelker, and Stefan Savage. 2019. *Measuring Security Practices and How they Impact Security.* In Proceedings of the ACM Internet Measurement Conference (IMC '19).

ABSTRACT OF THE DISSERTATION

Names to Conjure With: Measuring and Combating Online Adversaries
by Examining Their Use of Naming Systems

by

Audrey Randall

Doctor of Philosophy in Computer Science

University of California San Diego, 2023

Professor Aaron Schulman, Chair
Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair

Defenders combat online adversaries by understanding their behavior, the resources they depend on, and their strategies and tactics. However, measuring adversarial activity directly is often challenging, because adversaries take steps to obfuscate their behavior and evade detection by defenders. To overcome this challenge, defenders may leverage the knowledge that adversaries rely on licit, external resources, whose business models do not require secrecy. These resources may therefore leak valuable information, including the prevalence of threats, the relative effectiveness of competing adversaries, the strategies adversaries use, or the resources

and infrastructure they rely upon. Such information can help defenders prioritize threats and decide which components of an ecosystem to target for interventions. This dissertation presents a new framework for designing measurement techniques and interventions for online adversaries: I leverage the information leaked by *naming systems*. I show that because naming systems are both lists of an adversary's resources and critical resources themselves, observing them enables defenders to measure adversaries' prevalence, compare their harmfulness, analyze their infrastructure, and more, thus improving interventions by identifying the most effective resources to target and prioritizing the most dangerous threats.

I present four studies that each leverage some aspect of a naming system to measure an adversary's behavior and inform defenses against it. First, I measure the prevalence of overt stalkerware in the wild, by using privacy-preserving DNS cache snooping on four public DNS resolvers. Second, I determine the location in the network of DNS redirection attacks, by exploiting the format of certain special DNS responses. Third, I investigate the abuse of blockchain-based naming systems (BNSes) by malware operators, and design interventions leveraging BNS components to disrupt malware campaigns. Finally, I measure an emerging web privacy threat, UID smuggling, by participating in the naming system built by trackers to link user identifiers with behavioral data. In each case, I measure or design defenses against an adversary that would be difficult to study without examining the information leaked by a naming system.

Chapter 1

Introduction

Staging effective interventions against online adversaries — such as stalkers, trackers, malware operators, and perpetrators of fraud — requires a thorough understanding of all aspects of their behavior. An intervention is an action taken against an adversary to disrupt or curtail their activities by denying them some resource. However, not all interventions are equally effective. An intervention is most likely to be successful if the resource it targets is both indispensable and difficult for the adversary to replace. To identify such critical resources, a defender must understand the ecosystem the adversary operates in, their targets, and the details of their attacks.

Adversaries, however, have no interest in allowing defenders to study their behavior, and often take steps to conceal their activities. They are thus difficult to measure using straightforward techniques that work on similar but non-adversarial systems. For example, if a researcher wished to enumerate the users of a benign class of mobile applications, she might use download statistics from app marketplaces such as the Google Play or Apple store. If she wished to understand user experiences, she might find volunteers willing to respond to a survey. If she needed to reverse-engineer a distributed system (e.g., a blockchain), she might examine open source code, become a member of the system, or contact the maintainers to ask questions.

In contrast, studying adversarial applications and systems is more difficult. Malicious apps might disguise their presence on app marketplaces or require their users to side-load them. Their installers may be unlikely to admit their activities when questioned. Unlike benign

distributed systems, adversarial systems like botnets are unlikely to make their source code, membership in their systems, or their developers available to researchers. Thus, some creativity is required to develop novel techniques to measure adversarial systems.

However, no system, benign or otherwise, operates in a vacuum. Even the most stealthy of adversaries requires certain external resources to make their application work correctly. For example, a botnet operator might rely on an external hosting provider to host the botnet's command and control (C2) servers, public or ISP-run DNS resolvers to resolve the C2 server's name, and public network infrastructure to route botnet commands. While an adversary might choose to do without any of these external resources, implementing an alternative is often prohibitively expensive. For instance, if a botnet operator chooses to bypass a hosting provider and host their C2 servers on their own infrastructure, defenders could seize those servers and force the adversary to replace potentially costly hardware. In contrast, by using a bulletproof hosting provider, an adversary can take advantage of the economies of scale that allow bulletproof providers to run numerous machines, and easily swap their C2 server to another machine if defenders interfere with the original. Similarly, a malware operator could avoid using DNS to contact their C2 servers, perhaps by hard-coding the servers' IP addresses into deployed malware, to prevent defenders from taking down the domain. However, this strategy risks losing control of infected devices: if a defender seizes the C2 server or blocks its IP address, infected devices can no longer discover any new server's address, because they cannot contact the old one. In general, even if an adversary can eliminate one or more external resources, attempting to remove *all* of them is usually not cost-effective. Any remaining external resources present opportunities for defenders to glean more information about an adversary's behaviors and attack vectors.

In this dissertation, I focus on one specific type of resource: *naming systems*. I define a naming system as any system that maps names or identifiers to other information, such as IP addresses or metadata about entities within the system. I argue that out of all the components in an ecosystem, naming systems are often the most critical resource for defenders to understand, for two reasons:

- First, at a fundamental level, a naming system is a list of the resources an adversary controls.

- Second, the naming system itself may be one of the indispensable and scarce resources the adversary relies on, that a defender could intervene with to disrupt the adversary.

In this dissertation, I study naming systems in two ways: by analyzing their recorded names and records, and by observing the `get` and `set` requests that update those records in real time. The former approach can reveal how much physical infrastructure an adversary controls, how important certain servers are (which might tell defenders how to prioritize takedowns), the adversary's location or country of origin, and so forth. Additionally, dissecting how an adversary generated some of its names might allow a researcher to identify the rest from the set of all names, and seize them all simultaneously. Enumerating all recorded names in a system may also reveal how many are abused by adversaries versus used by benign actors, which might help defenders design interventions with lower collateral damage.

In contrast, observing which names are actually *requested*, as opposed to stored but potentially never accessed, can reveal both the popularity of the resources they represent and the utility of those resources to the adversary. Measuring the frequency at which resources are accessed can reveal which of several competing adversaries is most successful, which helps defenders decide which to prioritize. Additionally, observing which resources are requested over time can reveal how agile an adversary might be in evading any particular intervention. For example, if defenders learn that an adversary is capable of quickly recycling resources, they can avoid seizing names or infrastructure that will soon be replaced regardless of their efforts. Observing small details of request formats can also reveal valuable information about an adversary's tactics, such as their location in the network or clues about their identity.

In the following chapters, I will demonstrate why naming systems are such valuable resources for defenders and how I used them to study and combat specific adversaries. I will show that because naming systems are both lists of an adversary's resources and critical resources

themselves, observing them enables defenders to measure adversaries' prevalence, compare their harmfulness, analyze their infrastructure, and more, thus improving interventions by identifying the most effective resources to target and prioritizing the most dangerous threats.

To that end, I first performed a study that leveraged DNS requests to measure the prevalence of stalkerware. Stalkerware is a class of applications that exfiltrate a target's communications, location, or other sensitive personal information. Stalkerware is challenging to measure in the wild because it is intentionally extremely difficult to detect on a target's device. I pioneered a technique to measure its prevalence by leveraging DNS cache snooping on public resolvers. I reverse-engineered the caching architecture of four large public resolvers to detect when stalkerware apps "phone home." I demonstrated that despite their complex and opaque caching structures, public resolvers can be used to measure the prevalence of rare, harmful applications such as stalkerware. I also derived the first estimate of overt stalkerware prevalence in the wild: I measured a lower bound of nearly 6,000 simultaneous targets of stalkerware in the U.S.

While performing this study, I discovered that some of my DNS requests were getting transparently redirected to resolvers I had not intended to contact. The responses' origins were spoofed: they appeared to be the destination resolver I had specified, rather than the resolver that had truly answered the query. I discovered that some network operators were redirecting DNS queries to their own resolvers, presumably either to track the presence of malware on their networks or to snoop on user browsing habits. To gain insight into which entities might be responsible for DNS redirection, I pioneered a distributed measurement technique to detect where in the network redirection occurs. I used DNS CHAOS queries and queries to bogon IP addresses to determine whether redirection occurred at the Customer Premises Equipment (CPE), within the ISP's network, or at an indeterminate location.

While DNS is often the naming system used by malware operators, it has a significant disadvantage from a miscreant's point of view: DNS domains can be taken down ("sinkholed") by the registrars that sold them, rendering them inaccessible and useless. Domain takedowns

are a crucial type of intervention for defenders, who in the past have taken down entire botnets by sinkholing the domains of their C2 servers. Some malware operators have responded to this threat by adopting blockchain-based naming systems (BNSes), which are purportedly much more difficult to intervene with than DNS. My next study examined the ecosystem of blockchain-based naming systems to determine how impervious they truly are to the interventions that are possible with DNS. First, I studied the resources required for an infected host to access a blockchain-based naming system, and identified potential locations where defenders could intervene. Next, I performed a study of how existing BNSes are used in the wild to quantify the amount of existing abuse. I determined that while some types of BNSes are already heavily abused, others have surprisingly little detectable malicious activity. Finally, I enumerated the BNS names that leaked to a DNS root server over the course of several days, to determine not only which names are registered, but which names are actively requested in BNS systems. I concluded that defenders still have viable options for intervening with BNSes and that more modern, general-purpose BNSes are unlikely to become a threat until their cost-of-use decreases significantly.

My final study focused on the naming system built by web trackers to record user behavior across websites by linking it to global user identifiers (UIDs). Until recently, advertisers could build a database of user information where UIDs belonged to a single, global namespace. However, in the past few years, browsers have deployed defenses that effectively partition the namespace that trackers may use by partitioning the storage they are allowed to access. Instead of building a global namespace of UIDs by accessing global storage, trackers are restricted to accessing a different storage area for each website the tracker appears on. In theory, this defense should prevent trackers from assigning the same UID to a user across different websites, thus disabling their ability to connect information about a user's behavior across those sites. In practice, I find that a non-trivial number of trackers have simply started sharing UIDs across supposedly partitioned namespaces, linking them together to form an approximation of a global namespace and regaining their ability to track users across sites. I labeled this practice "UID smuggling," and measured its occurrence by designing a web crawler. I found that UID smuggling

was present on approximately 10% of the random navigations made by my crawler while visiting approximately 50K popular websites.

The remainder of this dissertation is structured as follows. Chapter 2 presents a high-level model of naming systems that describes how they can be used to measure adversaries and inform interventions. Chapter 3 describes my work on reverse-engineering the DNS caching architecture of large public resolvers, and my use of those caching models to estimate the prevalence of stalkerware and other rare forms of abuse. Chapter 4 presents my technique for identifying the network location of DNS redirection attacks, which can yield clues to the perpetrator's identity. Chapter 5 describes the ecosystem of blockchain-based naming systems and presents possible defenses against malware that abuses those systems. Chapter 6 documents my measurement of web trackers' attempts to reconnect the partitioned namespace they use to map IDs to user behavior information. Finally, Chapter 7 summarizes this dissertation, presents key takeaways from my work, and describes directions for future research.

Chapter 3, in part, is a reprint of the material as it appears in *Proceedings of the Internet Measurement Conference 2020*. Audrey Randall, Enze "Alex" Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in *Proceedings of the Internet Measurement Conference 2021*. Audrey Randall, Enze "Alex" Liu, Ramakrishna Padmanabhan, Gautam Akiwate, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in *Proceedings of the APWG Symposium on Electronic Crime Research (eCrime) 2022*. Audrey Randall, Wes Hardaker, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in *Proceedings of the Internet Measurement Conference 2022*. Audrey Randall, Peter Snyder, Alisha Ukani, Stefan Savage,

Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 2

Background

The primary goal of my work is to understand adversaries by examining their use of naming systems, and use that understanding as the basis for improving defenses against them. To that end, I must understand in great detail how each naming system works from both a logical and implementation-based perspective. This understanding is crucial for two reasons. First, when my goal is to measure an adversary's behavior, I must thoroughly model all of a naming system's components to determine which ones are the most likely to leak information. Second, although naming systems can be effective locations to stage interventions against some adversaries, performing such interventions requires overcoming various logistical, technical, and even political challenges. Specifically, interventions must be enacted by entities within the naming system that have the capability and motivation to do so, such as DNS registrars or proxies in BNSes. Identifying which entities can be convinced or legally compelled to aid defenders requires understanding the organization of the system. Interventions may also cause collateral damage to licit users of the naming system if not carefully designed. For example, taking down a proxy that serves both malicious and benign traffic can cause unexpected harm if defenders are unaware that many licit users utilize it as well.

While many aspects of naming systems are well-documented and well-understood, real-world complexities and constraints often lead these systems to work differently in practice than they do in theory. For example, when a query is redirected and answered by a different entity

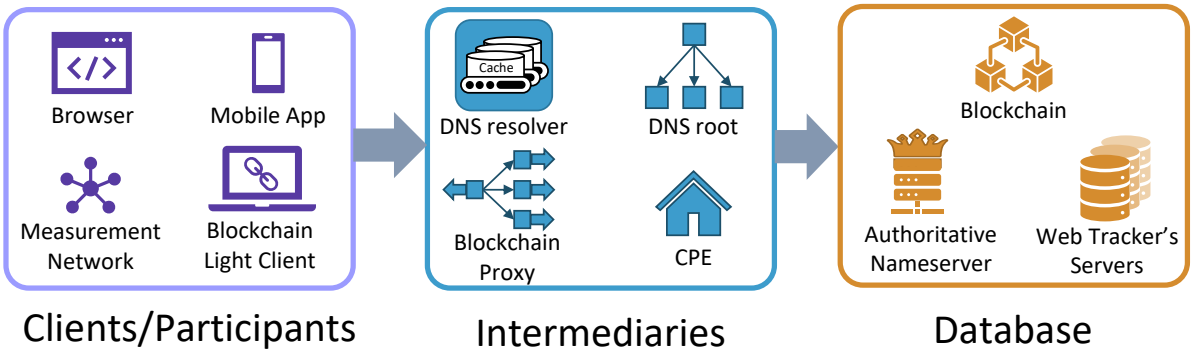

Figure 2.1. Physical architecture of a naming system in the general case.

Table 2.1. Keys and values in different naming systems.

Naming System	**Hierarchical?**	**Key**	**Selected Values**	**Example of (key: value)**
DNS	✓	Domain	IP address	(example.com: 93.184.216.34)
			Text	(id.server: res721.burr.rrdns.pch.net)
BNSes	✓	Name	Wallet address	(alice.eth: 0x2805A...)
			Website URL	(john.crypto: https://ipfs.io/...)
			Text string	(weedlocator: "hello fx-wallet")
			IP address	(namecoin.bit: 91.219.237.223)
UID Namespace	×	User ID	*User profile*	(D55FCE51...: *[behavior profile]*)

than the intended destination, the misformatted response may reveal both the presence of the redirection and the identity of the redirector (Chapter 4). Furthermore, there may be no public documentation describing a naming system in sufficient detail to reveal where information about adversary behavior might leak, which forces defenders to perform measurement studies instead. However, defenders are aided in designing these measurement studies by the similar logical structure that many naming systems share, despite the differences in their implementation details. This chapter describes how naming systems work from both an abstracted, high-level view and from a detailed, implementation-based perspective.

2.1 Naming systems in the abstract

Naming systems are frequently designed to abstract away complexities from their end users, which allows clients to view them semantically as simple databases of key/value pairs. Depending on the naming system's purpose, keys (names) may map to small amounts of information, such as IP addresses, or larger amounts, such as collections of user behavior traits. Taking terminology from DNS, I refer to a key/value pair in any naming system as a "record" in this dissertation. End users interact with the naming system by making `get` and `set` requests. While only certain users (usually the owners of names) have the authority to `set` records, any participant may generally make `get` requests. Table 2.1 shows examples of keys and values in each of the naming systems I study. In DNS, keys are domain names; in each of the BNSes, keys are simply names, and in the UID-based naming system, keys are UIDs. Keys in each naming system may have different types of records: some of these are detailed in Table 2.1.

However, the user must first know which names to request, and many naming systems do not make their full databases publicly available. For example, database records in DNS (called "zone files") are generally obtainable for Top Level Domains (TLDs), but not for Second-Level Domains (SLDs), which are usually proprietary. A critical challenge therefore lies in discovering which names adversaries are using without access to the entire database. While this is not trivial, a defender may often reduce the scope of this problem by leveraging a certain property of some naming systems: *hierarchical structure*. The records in a naming system form a namespace, which may be either flat or hierarchical. Hierarchical namespaces provide a specific advantage for a researcher: if she identifies one name that an adversary uses, all of that name's "subordinate names" (names whose parent in the hierarchy is the name of interest) likely belong to the adversary as well. The majority of the namespaces I cover in this dissertation are hierarchical; only one is flat (Table 2.1).

Table 2.2. Participants in the layers of each naming system I study.

Naming System	Clients	Intermediaries	Database
DNS	Stalkerware, browsers, mobile apps	Public resolvers, root servers	Authoritative nameservers
BNSes	dApps, browsers	Proxies, bootstrap nodes	Blockchains
UID Namespace	Tracking scripts	n/a	Trackers' servers

2.2 Implementation details of naming systems

Defenders can obtain valuable insights by requesting and analyzing the information in the database, such as which physical resources the adversary is using, which adversaries may be colluding or controlled by the same operators, and more. However, using this technique alone, a defender cannot tell which names are used frequently in the wild or how often they are requested; for this purpose, a defender must observe the adversary's workload that uses the database. These metrics can reveal further information that is valuable to defenders, such as how widespread an adversary's attack is or which of several competing adversaries has the most widespread attacks. Answering these questions is not often possible unless the defender can observe the adversary's `get` or `set` requests, which is usually not straightforward. Observing requests directly often requires controlling naming system infrastructure or having access to a multitude of vantage points within the system. My work demonstrates that it is possible to instead leverage *leaked information*: information that is not intended to be public, that nevertheless reveals aspects of an adversary's behavior. I reverse-engineer each naming system in detail to discover which components or implementation aspects inadvertently reveal valuable data.

Reverse-engineering a naming system may sound simple in principle, because from the perspective of its users, it is often a simple key/value store. However, the implementation details of these systems are by necessity more complex. Considerations such as scalability, performance, resource constraints, and federated management strongly influence naming system

implementations. In this dissertation, I show how a defender can exploit these design decisions to infer how an adversary (or any participant) uses the system.

In this dissertation, I represent the implementation and deployment of a naming system in the general case using three layers (Figure 2.1): (1) the clients or participants of the naming system, (2) zero or more intermediaries (e.g., resolvers, proxies, etc.), and (3) the database. Concrete examples of the participants in each layer are detailed in Table 2.2. Clients originate `get` and `set` requests to the database. I observe and imitate the clients of adversaries to gather the set of names they use. By observing clients directly, I can establish that the requested names truly belong to that adversary. For example, when I participated in the naming systems used by web trackers to gather user information, I was able to record the UIDs (names) that were set by tracking scripts (the clients of that system). In a separate project, I recorded the DNS requests of stalkerware apps to gather the names of their servers. I also acted as a client myself, by making DNS requests and observing how they were redirected in transit, and by requesting blockchain names and recording the resolution process on various blockchain-based naming systems. However, while studying clients is often easier than gaining access to intermediaries or databases, each client offers only one vantage point into the system. Thus, certain questions centered around workload (such as how often a particular name is used) are better answered by studying other naming system components.

Intermediaries in a naming system can play several roles: they might cache information to reduce request time, know which piece of a distributed database contains a particular record, translate web protocols to the naming system's own protocol, and more. From a measurement standpoint, an intermediary is a centralized vantage point that observes requests from many different clients. Intermediaries are often distributed geographically to reduce the latency of client requests. Consequently, a defender may leverage this distribution to gain insight into the behavior of an adversary in particular locations, because each instance of an intermediary communicates with the clients "closest" to itself (either geographically or in terms of network topology). Observing the frequency of requests at intermediaries may reveal the prevalence of

an attack, for example by allowing a researcher to enumerate infected devices. Intermediaries may also inadvertently leak information through details of their implementations. For example, the format of responses to certain DNS queries may reveal who controls the DNS resolver that answered the query, and thus which entity is meddling with DNS requests. Finally, from the perspective of a defender planning an intervention, intermediaries are bottlenecks that may be willing and able to filter an adversary's requests. For example, in the case of BNSes, I observe that certain intermediaries (such as proxies) are the only entities in the system that have the ability to stage interventions.

The database stores the naming system's records. Analyzing the set of all records can reveal how participants plan to use the system in the future, what resources (e.g., servers) they control, who controls which resources, or which adversaries are colluding. Some databases, such as those stored by BNSes, are public, but many others are not. When the database is public, an observer can simply search through it and attempt to identify names owned by adversaries. When it is private, the names of interest must be identified using other means. I present several techniques for finding adversaries' names, including examining names that "leak" outside a naming system (Chapter 5), running and analyzing adversarial applications (Chapter 3), and posing as an adversary's target to observe its behavior (Chapter 6).

Chapter 3

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers

3.1 Overview

The goal of this study is to measure the prevalence of stalkerware, because at the time of writing, no comprehensive estimates existed of how many people are targeted by stalkerware in the wild. To complicate the issue, the term "stalkerware" is an umbrella that encompasses two classes of application: *dual-use stalkerware* is nominally benign software that can be repurposed for stalking, such as "find my phone" or anti-theft apps, while *overt stalkerware* is advertised and designed specifically for tracking a target. Dual-use stalkerware has been found to be far more common than overt stalkerware, but prior work in this space has primarily been limited to clinical settings [105, 95, 61]. By the time survivors seek help at a clinic, they have often taken steps to remove potential stalkerware such as performing a factory reset on their devices. While the efficacy of this strategy in general has not been studied, all of the overt stalkerware that we examined will indeed be removed by a factory reset. Thus, clinics may be unable to determine whether overt stalkerware was ever installed on a survivor's device.

To measure the prevalence of overt stalkerware in the wild, we leverage stalkerware's use of a naming system — DNS — to exfiltrate target data to centralized servers. We examine information leaked by an intermediary: in this case, public DNS resolvers. In the process, we expanded our technique to cover other types of rare abuse that are difficult to measure or

under-studied. While there is a range of approaches to measure the prevalence of widespread abuse (e.g., spam [174]), characterizing the amount of rare abuse—where a small number of users experience or cause a significant amount of harm—has remained elusive. While harmful Internet behavior manifests in many different forms, using different protocols and on different platforms, virtually all depends on DNS.

While in most cases, it is not possible as a third party to directly measure the number of global DNS queries for a given name, we can infer them indirectly using DNS cache snooping: a technique that probes DNS resolvers to observe if a domain is in the cache, therefore implying that a user must have previously accessed it.

In this work, we focus on techniques for cache snooping large public DNS resolvers. Due to their scale, public resolvers both provide large-scale measurement opportunities and, due to their aggregation, sidestep some of the traditional privacy concerns of cache snooping. For example, as of May 2020, APNIC's DNS service popularity measurements indicate that $\sim$10% of web users appear to use Google Public DNS (GPDNS) as their primary DNS resolver [140], while Cloudflare and OpenDNS each serve $\sim$1% of web users.

However, public DNS resolvers consist of many independent caches operating in independent Points-of-Presence (PoPs), which makes them among the most challenging DNS resolvers to cache snoop. Through controlled experiments, we infer the load-balanced multi-layer distributed caching architectures of the four most popular resolvers. To the best of our knowledge, we are the first to model the behavior of these caching architectures and how they relate to user accesses. Building on these models, we demonstrate that it is possible to snoop public DNS PoPs and estimate how many caches contain a specific domain. Surprisingly, we found that GPDNS appears to dynamically scale the number of caches that contain a particular domain name based on the number of users accessing it; we observed up to several thousand uniquely identifiable caches for one domain name (Section 3.5). This behavior is a likely explanation for the unusual caching behavior of GPDNS that was reported, but not explained, in prior work [184, 189].

We present Trufflehunter, a tool to snoop the caches of public DNS resolvers. We

evaluate the accuracy of Trufflehunter's cache behavior modeling with a large-scale controlled experiment. Our relative error in estimating the number of filled caches for each resolver varies from 10% to −50%, with the exception of one unusual Cloudflare location where our error is 75% (Section 3.5). Trufflehunter's error varies depending on the caching architecture of the resolver: it can estimate the cache occupancy of OpenDNS and Cloudflare more accurately than Quad9 and GPDNS. This error may seem large, but because Trufflehunter consistently *underestimates* cache occupancy, it can provide a *lower-bounded* estimate of the prevalence of rare user behaviors (Section 3.6).

We demonstrate Trufflehunter with several case studies on abusive Internet phenomena. We found that some of the most concerning smartphone stalkerware apps have a minimum of thousands of simultaneous active users. We also found academic contract cheating of the services were significantly more popular than the others, and their popularity wanes during the summer.

Trufflehunter is open source and available at:

https://github.com/ucsdsysnet/trufflehunter

3.2 Background

In this section, we describe how we can measure the prevalence of rare Internet user activity by probing the caches of public DNS resolvers. We begin by describing why public DNS services have become a key vantage point for observing uncommon behavior of Internet users. Then, we describe how the complex caching architecture of these services makes it possible to externally measure the minimum number of simultaneous users that have queried for a domain name. Finally, we outline how this complex caching architecture makes it challenging to accurately estimate the number of users that accessed a domain with cache snooping techniques.

3.2.1 A variety of users query public resolvers

Initially, the set of users that adopted public DNS were Internet power users who were privacy and security conscious. However, public DNS is now becoming a popular default

configuration on networks and in software. This trend has caused a wide variety of users to adopt these services; indeed, many public DNS users today did not explicitly configure their devices to use public DNS. The adoption of these services is largely driven by two factors: (1) network operators and equipment vendors configuring networks and devices so that users default to using public DNS services as their primary or secondary resolver, and (2) developers hard-coding public DNS resolution into their software.

Enterprise network administrators have switched from running their own DNS resolvers to pushing users to public DNS resolvers because that can improve reliability [108]. Small-scale ISPs have also switched to public DNS to avoid the operation and maintenance cost of providing their own DNS resolver. For instance, GPDNS has a formal vetting process where ISPs can request to remove GPDNS's rate limits so they can have their entire customer base use GPDNS as their primary resolver [98]. Also, public DNS is often adopted by administrators and ISPs because it provides additional security measures for their users. For example, Quad9 and OpenDNS both block DNS requests for domains that are reported to be malicious on threat intelligence feeds [145, 178]. This security feature was reported to be the primary reason the NYC city-run public WiFi network switched to using Quad9 as its default DNS resolver [177]. Security is also cited as the primary reason that enterprises and schools have switched to using OpenDNS [165]. Public DNS resolvers have also been set as the default resolver in networking equipment as a means of improving performance. The most notable example of this trend is the "Google Home" WiFi router, which ships with its default configuration to resolve all DNS queries with GPDNS [91].

Software developers have also increased public DNS adoption by hard-coding their software to send DNS requests to public resolvers. The most notable instances are because public DNS services offer the latest DNS security features. For example, when Firefox deployed the privacy protections provided by DNS-over-HTTPS, it hard-coded Cloudflare's public DNS resolver as the default resolver for all of their U.S. users [74]. Additionally, the reliability, and wide availability, of public DNS makes it a common choice as a hard-coded backup resolver.

For example, in 2017 Linux distributions started shipping with GPDNS hard-coded as a backup resolver in SYSTEMD [153, 196], and in 2019 added Cloudflare as well [133].

3.2.2 Complex caching can reveal many active users

We now describe how cache snooping public DNS resolvers can provide a lower bound on the number of users accessing domain names—without revealing who has accessed these domain names. Public DNS resolvers do not operate as a large contiguous cache with global coverage. Rather, they operate many fragmented DNS caches [30, 217, 158] and load-balance queries across many caching resolver instances [129, 192].[1] This architecture can have a negative effect on their performance: even if a user recently resolved a domain name with a public DNS resolver, subsequent requests to that domain may not be serviced from a cache. However, this performance limitation is also an opportunity for establishing a non-trivial lower-bound on the number of users accessing a domain.

To demonstrate how cache snooping public DNS services can reveal a non-trivial number of users, we start by explaining how a typical query is cached in a public DNS resolver's hierarchy (Figure 3.1). In particular, we focus on the three steps used to resolve a query on a public resolver, where the response to the query can be cached in one of many independent caches. This cache architecture is a generalization of how all of the large public DNS resolver caches that we study operate. Section 3.3 later describes the details of how caching works in each of the resolvers.

① Users direct their query to one of the public DNS service's PoPs by sending the request to one of the service's anycast IP addresses [158]. These addresses are announced by routers in PoPs distributed geographically around the globe.[2] This anycast DNS architecture is similar to the anycast load balancing that the root DNS servers use [185, 155, 50, 57]. In our experiments we found that, for large resolvers, PoPs operate their caches independently (Section 3.5).

[1] GPDNS served 400 billion responses per day in 2015 [100].

[2] As of May 2020, GPDNS has 33 PoPs worldwide, Cloudflare has 46 PoPs in the U.S., Quad9 has 27 PoPs in the U.S., and OpenDNS has 11 PoPs in the U.S. & Canada.

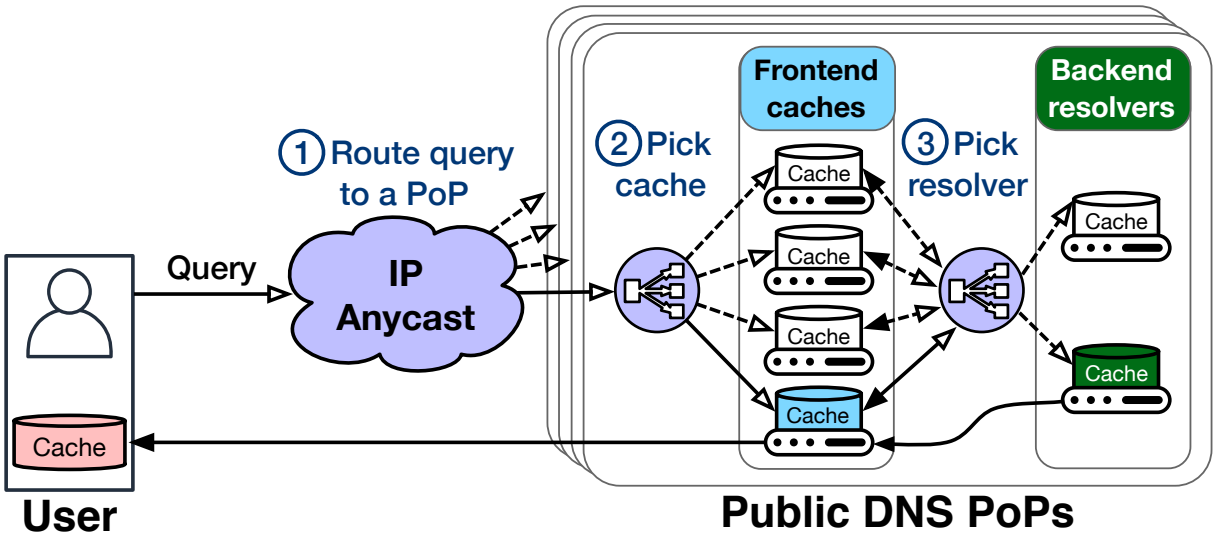

Figure 3.1. A user's DNS query is handled by a set of load-balanced front-end caches that are backed by a set of load-balanced backend resolvers.

② Then, within a PoP, a query is load-balanced to a pool of frontend caches for the backend caching DNS resolvers [73]. The load balancer selects from the pool of frontend caches based on a policy that distributes the load across these caches. These frontend caches can be isolated, creating the possibility of having more independent locations users' query responses can be cached.

③ If the selected frontend cache does not have an entry for the domain name, the query will be forwarded to one of a pool of backend resolvers via a second load balancer. Backend resolvers operate independent caches, introducing yet another opportunity for multiple users to have their queries cached independently.

After the backend resolves the domain name, all of the caches along the path in the hierarchy are filled with the response. First, the backend resolver fills its cache, and in some cases responds directly to the user. Then the frontend resolver fills its cache, and finally the response is sent to the user, which fills its local cache. The presence of user-local caches makes it easier to count the number of users accessing a rarely-used domain name: it effectively limits the number of cache entries that can be created in a public DNS resolver to one per user at a point in

	Domain name	Recurse?	Timestamp (s)	TTL (s)	Result
Cache hit	www.example.com	No	1591002798	60	93.184.216.34
Cache miss	www.example.com	No	1591002825		

Figure 3.2. Examples of DNS responses for a cache miss and cache hit. The cache hit has a TTL in the reply, the cache miss does not.

time. All major operating systems operate have DNS caches [35, 128, 149, 31, 161], including MacOS, Windows, and Linux. Additionally, many browsers operate their own local cache such as Safari, Chrome, and Firefox. Some home networks and organizations also run a caching forwarding resolver for all users on their entire network. These local caches will effectively limit a household or organization to filling only one cache in a public resolver, per domain, at any point in time.

Snooping Public DNS caches can provide a lower-bound on the number of active users

If we can estimate how many of these caches hold a particular domain at any point in time, we can obtain a lower-bound on the number of simultaneously active users of that domain. However, this estimate is strictly a lower-bound on the number of users that may have requested this domain—we cannot observe how many users had their query serviced by one of the caches. As we are limited to counting only one user per cache, snooping will be most useful for estimating the popularity of *rare* domains. It will not provide any new information about the prevalence of domains that are already known to be popular.

Does DNS security limit counting users with cache snooping?

DNS security and privacy technologies, such as DNS-over-HTTPs, DNS-over-TLS, as well as DNSSEC, do not change the cache model for DNS described above. All queries made with these protections enabled will be served out of the same caches as insecure queries.

3.2.3 Public DNS cache snooping challenges

The complex multi-level caching hierarchy makes it feasible to estimate a non-trivial number of active users of a domain name. Unfortunately, it also makes it challenging to accurately estimate the number of caches that have been filled. Cache snooping a resolver with a single cache—as has been investigated in prior work—is straightforward. The most direct way of snooping a cache is to "probe" it by making a query for a particular domain name with the Recursion Desired (RD) flag unset. Unsetting Recursion Desired prevents the backend resolver from doing a recursive query to get the uncached answer, causing it instead to report a cache miss by not including a result in the answer. The DNS response from a cache probe contains limited information (Figure 3.2) that answers the following questions: is the domain name currently cached (indicated by the existence of a DNS Answer section having a nonzero TTL), and how long has it been cached (as inferred from the response timestamp and the remaining TTL in the response)? Cache snooping a large public DNS service to measure how many caches are occupied is significantly more difficult because each probe (non-recursive DNS query) returns information about only a single cache in the resolver.

From the limited information available in these single-cache responses, we somehow need to determine how many independent caches have been filled at the resolver. Note that each cache probe will provide the status of only one cache, in one PoP: we do not know what cache, nor do we know what PoP, the response came from. Additionally, we cannot tell if the response came from a frontend cache, a backend cache, or both (if they are synchronized). We also need to consider what happens when a cache entry is shared between caches. For instance, is the same TTL value stored in the frontend cache when copying from a backend resolver? What about when cache entries are shared between frontend resolvers? We will show in the next section that the data in the simple cache probes described above (Figure 3.2) is sufficient to estimate how many caches a domain is in at one time.

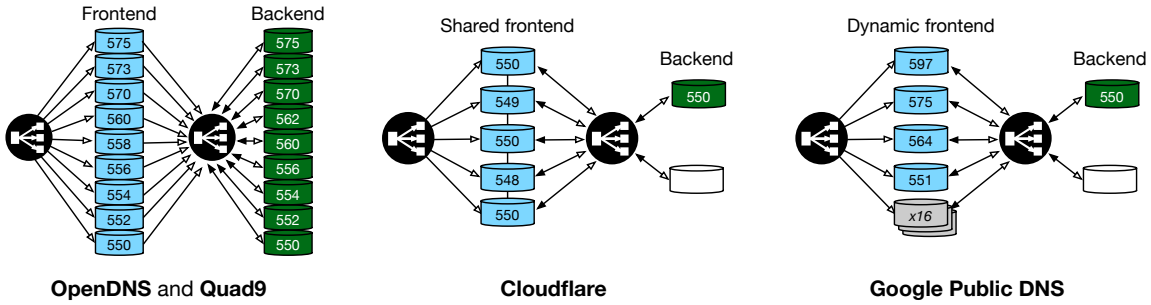

Figure 3.3. Inferred cache architectures of the four largest public DNS services (50 secs after filled with 600 sec maximum TTL)

3.3 Snooping Public DNS Caches

Cache snooping public DNS resolvers with Trufflehunter is possible because it can interpret the multi-level caching architecture's behavior by observing DNS responses. Specifically, it sends a collection of cache snooping probes (non-recursive DNS queries) for a domain name towards a resolver and deduces what the responses reveal about how many caches are occupied by that domain name. Different resolvers can use widely different architectures, however, introducing resolver-specific challenges in interpreting responses to cache snooping probes. In this section, we describe how we analyzed and modeled resolvers' caching architectures. We also show how Trufflehunter's inference technique will be tailored to each resolver's architecture.

In this section, we describe the cache inference logic we built in Trufflehunter based on our observations about how the caching architecture operates for the four largest public DNS resolvers. We explain how we inferred their behavior with a combination of controlled experiments—where we intentionally put a domain name in as many caches as possible in each resolver—and public information released by the resolvers. Each of the architectures presents their own opportunities or limitations for observing a non-trivial number of independent caches that have the same domain. We discovered that some resolvers employ caching strategies that allow us to measure a large number of user accesses, but others employ strategies that impose

significant limitations on the number of caches we can observe. They also all present unique challenges that make it difficult to estimate the number of occupied caches with cache snooping.

Inferring behavior of DNS cache architectures

We infer the caching architecture of public resolvers by inserting DNS responses into as many caches as possible, and observing how often new cache entries appear to be made.

Cache-filling experiment

We send recursive queries for a unique domain name to one PoP of each public resolver once every two seconds. These queries were made by a single machine in AS 7922 (Comcast). Since our goal is to fill as many caches as possible, including the backend recursive resolver caches, we issue these queries with the Recursion Desired (RD) flag enabled. We controlled the domain name used in the experiment, allowing us to verify that certain responses were serviced by a backend recursive resolver. The behavior of the resolvers during this experiment will be similar to how the resolver's caches will look when a resolver has a constant stream of users requesting a domain name.

The data collected during this experiment are DNS responses from the resolvers. When resolvers operate independent caches, the primary indicator that a response is coming from a particular cache is the TTL in the response. We know that one of our queries caused a cache to be filled when the response contains the maximum TTL (i.e., the TTL returned by the authoritative nameserver). We know that a query was serviced from a cache—and therefore did not fill a new cache—if the TTL in the response is lower than the maximum TTL.[3] The TTL also reveals which cache the query was serviced from because TTLs of cached responses decrement one second per second: responses that were received N seconds apart, and also have a difference in their TTLs of N seconds, can be assumed to come from the same cache. Our observations of the pattern of TTLs in the responses form the basis for our technique (described in Section 3.4) to measure

[3]Most DNS resolvers age the TTLs of cached DNS responses once every second.

the number of filled caches with DNS cache snooping probes (i.e., repeated *non-recursive* DNS queries).

TTL Line

Responses that originate from the same cache should fall on a line with a slope of -1 (since TTLs decrease once per second) on a graph of TTLs over time. We use the term "TTL line" to refer to this line. TTL lines originate from a point in time where we infer that a cache was filled because we observe a response with the maximum TTL (600 seconds in our controlled experiment).

Visualizing DNS resolver caching behavior

The results of this experiment are presented as follows. For each resolver, we plot a point for every DNS response we receive during the experiment. The x-value is the time the response was received, and the y-value is the TTL contained in the response. We also draw a TTL line each time we observe a response with the maximum TTL. We only plot the first 50 seconds of each experiment because that is sufficient to show the general caching behavior. To make it easier to understand what cache architecture is producing this behavior, and to provide a visual comparison between the architectures, we show the states of the three different cache architectures at the end of the 50-second period in Figure 3.3.

3.3.1 OpenDNS and Quad9

OpenDNS and Quad9 presented the most intuitive caching behavior of the public resolvers. They both appear to be operating independent frontend caches (Figure 3.3). This architecture means that Trufflehunter can observe at most N_b simultaneous active users of domain names, where N_b is the number of backend resolvers operating at a PoP.

Figure 3.4 depicts the results of the cache-filling experiment that demonstrates this behavior (we omit the plot for Quad9 because its behavior is similar). As OpenDNS received repeated queries over time, we observed nine responses with the maximum TTL (indicated by

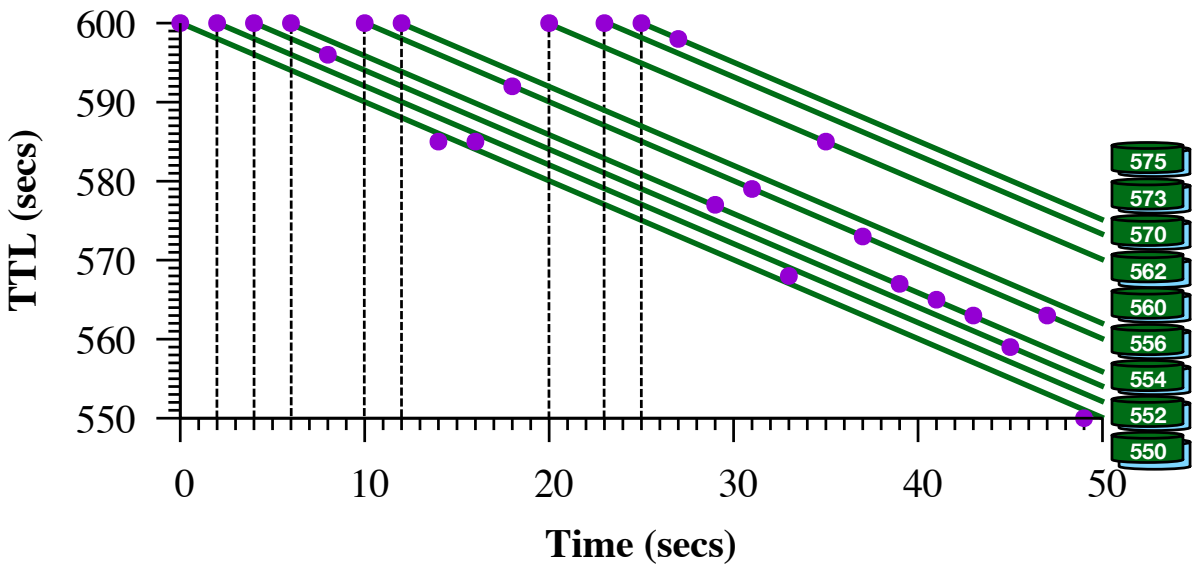

Figure 3.4. OpenDNS and Quad9 caching behavior

the vertical dotted lines). For each of these responses, we observed a query to the authoritative resolver. Therefore, we can conclude that a frontend cache did not have the entry cached, and the query was resolved by a backend resolver. All of the other responses that we received from OpenDNS had a TTL that fell on one of the TTL lines that originate from these nine responses with the maximum TTL (there is an inherent error of $+1, -1$ seconds that we address in Section 3.4.2). This behavior indicates that when a frontend cache does not have an entry, it copies the TTL from the response it gets from forwarding the query to a backend resolver, even if the backend resolver answers the query from its cache.

Estimating OpenDNS and Quad9 cache occupancy

Estimating the number of domain users active on Quad9 and OpenDNS requires estimating the number of independent backend resolvers that have the domain cached at any point in time. Recall that each TTL line in the recursive responses corresponds to one backend resolver having the domain name cached. Therefore, Trufflehunter can estimate this quantity by sending repeated cache probes for the domain, and counting the number of unique TTL lines it observes.

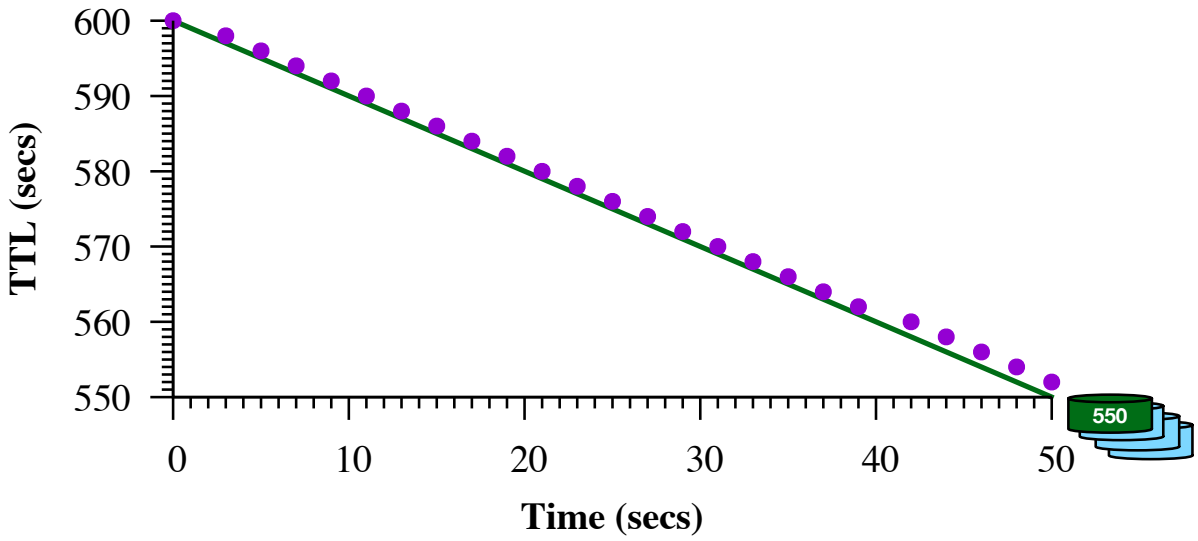

Figure 3.5. Cloudflare DNS shared caching behavior

3.3.2 Cloudflare DNS

Cloudflare's DNS service is the only DNS resolver we evaluated that operates a shared frontend cache architecture, as shown in Figure 3.3. Specifically, it uses knotDNS's resolver, which has a shared backing database for its frontend caches (e.g., memcached) [101]. This architecture means that, unfortunately, the lower-bound of number of users accessing a domain on Cloudflare will be very conservative—at most one user within a TTL interval at a PoP. However, for domain names that are infrequently used, such as the ones we design Trufflehunter to observe, this limitation is not significant. Additionally, domains often have short TTLs and Cloudflare operates its resolvers from numerous PoPs, allowing us to provide meaningful lower-bound estimates.

Figure 3.5 shows the results of the cache-filling experiment for Cloudflare DNS. The recursive queries to Cloudflare all produce responses that fall on one TTL line that originates from the time the first query was made. However, this one TTL line is not perfect: it slowly deviates from a slope of -1. We believe this deviation is due to the errors in the TTL that accumulate as the frontend resolvers copy the cached response from the shared cache into their

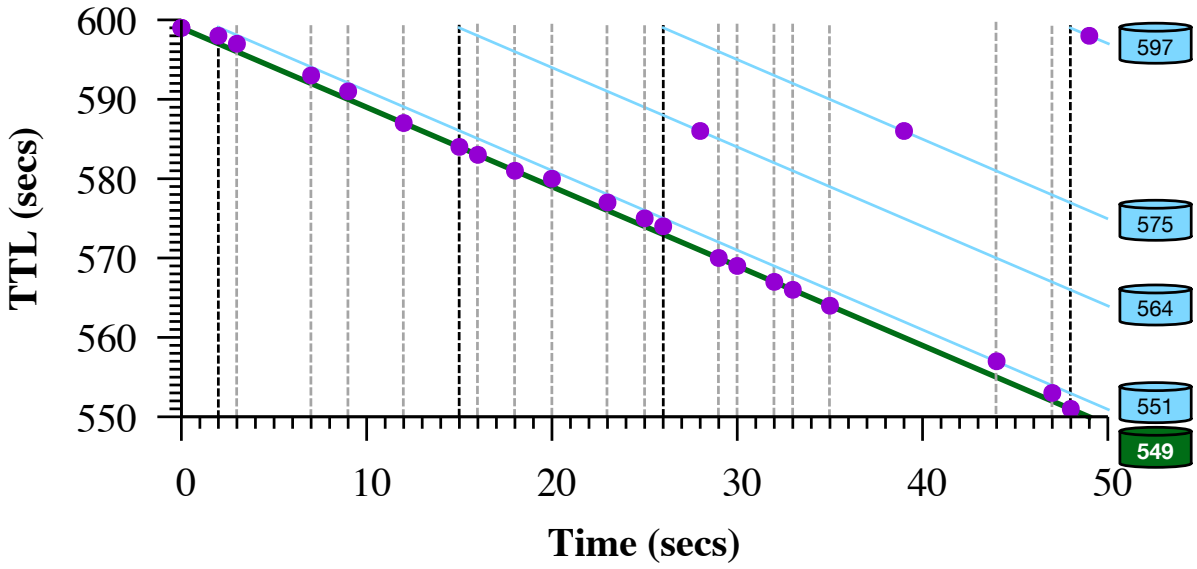

Figure 3.6. GPDNS's dynamic caching behavior.

local cache.

Estimating Cloudflare cache occupancy

All Cloudflare resolvers in a PoP share one cache. They also slowly drift away from the "true" TTL line. As a result, it is not sufficient to simply send cache probes and count how many individual TTL lines we observe. Drift in TTL of cache responses will effectively extend the amount of time that a result resides in the cache, ultimately longer than the maximum TTL. If we only sample the cache infrequently, and at irregular intervals, we may conclude there are more user requests than there truly are.

Instead, Trufflehunter counts how many times the cache was filled by applying a peak-finding algorithm that will find the points where all caches were empty and then filled again (details in Section 3.4.2). It also only allows one of these peaks per maximum TTL of the domain name to account for any spurious peaks that may be due to errors from cache sharing.

3.3.3 Google Public DNS (GPDNS)

Google's public DNS resolver has a unique caching behavior that enables counting a large number of users that are actively requesting a domain name. Like other services, Google describes their caching architecture as load-balanced frontend caches and backend resolvers [97]. However, when a query hits a frontend cache that does not contain the domain name, GPDNS appears to create a new, independent frontend cache. This behavior means that it may be possible to count nearly all accesses to a domain name by cache snooping GPDNS's frontend caches. We suspect the presence of so many frontend caches is a design choice to scale the caching infrastructure dynamically based on the number of requests to a particular domain—and may be a reasonable way of protecting their infrastructure from DDoS attacks. The cloned caches are deleted when their "parent" backend cache expires.

Figure 3.6 shows the unique results of the cache-filling experiment for GPDNS. The initial query is recursively resolved by a backend resolver, and several future responses appear to be serviced from the same cache because they follow the same TTL line. Strangely, several of the responses contain a TTL that does not correspond with that initial TTL line, and they also do not have the maximum TTL (indicating a new cache has been filled). These caches also do not appear to be backend resolvers because we only see one request to our authoritative server per maximum TTL epoch. Previous work also noticed these caches,[4] but could not explain where they came from [184, 189].

This behavior appears to be a result of the unique way that Google shares query results from backend resolvers with frontend caches. When a query is load balanced to a frontend cache that does not have the domain name in cache, it will forward the query to a backend resolver. The backend resolver will then do the same thing as other public DNS resolvers: namely, send the response to the query to the user, then it will fill the frontend cache that was empty. The exact way that they fill the cache, however, is unique: the backend cache will fill the *maximum*

[4] One paper referred to them as "ghost caches" [184].

TTL ($TTL_{max} - 1$ in the case of GPDNS [75]) in the frontend cache, rather than filling it with the current TTL of the cached entry at the backend cache. Therefore, each cache miss to a frontend cache fills a uniquely identifiable new frontend cache. We depict this behavior in Figure 3.6. Each cache hit to the backend resolver is marked with a vertical dotted line that marks that a frontend cache was filled at that time instant with the maximum TTL. When we draw TTL lines starting at these lines, we see that indeed four of them intersect the points that came from previously unexplained caches.

These cache entries may seem problematic because they effectively lengthen the duration that a domain entry is cached past the maximum TTL from the authoritative server. Fortunately, we observed that these cache entries are deleted as soon as the cache entry expires at the backend resolver.

Estimating GPDNS cache occupancy

While this cache-filling strategy creates an opportunity to count as many users as there are frontend caches, it also makes it difficult to cache snoop. The problem is that GPDNS fills a new frontend cache regardless of if the query is from a user (i.e., recursive) or a cache probe (i.e., non-recursive). Fortunately, it is possible to distinguish the caches created by cache probes from caches filled by users. Essentially, we account for all TTL lines that would be created by Trufflehunter's cache probes. When we observe a probe response that has a TTL on one of these lines, we simply discard it. Note that a consequence of this approach is that the more frequently we probe, the more frontend caches we may fill, and therefore the fewer users we can observe. To address this problem, we probe infrequently (i.e., five times per minute) relative to the duration of the maximum TTL (e.g., ten minutes), rather than probing as fast as possible.

Summary

This experiment demonstrates that public DNS resolvers often operate more than one independent cache in each PoP, creating the opportunity to estimate the number of users resolving a domain. We also describe how, by analyzing the TTLs obtained in DNS cache snooping, it is

possible to count the number of occupied caches. In Section 3.5, we evaluate how well these cache snooping techniques work with a controlled experiment across most of the U.S. PoPs of these providers.

3.4 Methodology

In this section, we describe the details of the probing and analysis methodology Trufflehunter uses to observe all unique TTL lines for a specific domain. First, we describe how Trufflehunter probes public resolvers at multiple PoPs using CAIDA's distributed Archipelago infrastructure [53]. Then, we describe the technique Trufflehunter uses to count unique TTL lines from these probe responses. Combined with the cache behavior models described in the previous section, Trufflehunter produces estimates of the number of cached copies of a domain across many of the PoPs of a public DNS resolver.

3.4.1 Probing multiple PoPs

Since DNS queries to public resolvers are routed using anycast [54], we have multiple opportunities to improve our estimates of the users of a domain. As each PoP implements multiple levels of caching, and each of these caches in turn can be probed for a domain of interest, each additional PoP we probe can significantly increase the lower bound on the number of users Trufflehunter can observe.

Ark enables Trufflehunter measurements across many PoPs

Measuring many domains over multiple PoPs requires a geographically distributed measurement infrastructure that offers considerable flexibility in the number and type of DNS measurements that it can make. In this study, we focus on the U.S. due to the diversity of PoPs used by public resolvers in the country. We considered three choices from which we could host Trufflehunter—RIPE Atlas [198], public clouds such as Amazon AWS, and CAIDA's Archipelago (Ark) project [53]—and chose the Ark network to run our measurements since it

offers diverse vantage points and the flexibility to implement and run continuous, longitudinal measurements. Though RIPE Atlas probes are more numerous and can contact more PoPs than the Ark nodes, more restrictions apply to their use; consequently we used Atlas probes only as controlled users for the smaller-scale experiments described in Section 3.5. We also considered using AWS, Google Public Cloud, and Microsoft Azure, but found that the Ark nodes have considerably wider coverage of PoPs. We deployed Trufflehunter on 43 Ark nodes distributed across the U.S.

DNS location requests identify which PoPs probes route to

While the existence of multiple PoPs per resolver enables improved measurement capabilities, it also introduces a layer of complexity: we need to identify to which PoP Ark nodes' DNS queries are routed. Doing so enables our analyses of filled caches for a domain at the per-PoP level (Section 3.5) and subsequent aggregation per resolver (Section 3.6). Fortunately, all four of the largest public resolvers provide ways to determine to which PoP requests are routed. Google makes the locations of GPDNS resolvers available in the form of a TXT record for the domain "`locations.publicdns.goog.`" Querying this record gives a map of resolver IP addresses to three-letter location codes. Another TXT record available at domain "`o-o.myaddr.l.google.com`" returns the non-anycast IP address of the resolver answering the query, which can then be looked up in the map. Similarly, TXT queries to "`debug.opendns.com`" return the three-letter airport code of the PoP that the query routed to in the answer. Quad9 and Cloudflare similarly make their locations available via CHAOS TXT queries to the domain "`id.server`" [118, 186]. For all experiments in the remainder of the paper, we identify PoPs by the three-letter code of a nearby airport.

Data collection

Our deployment of Trufflehunter on Ark nodes continuously runs two sets of DNS measurements from each node:

1. Given a list of domains to search for, Trufflehunter performs a DNS request for each domain five times per minute towards each public resolver we study. These requests are made with the RD flag unset (non-recursive). We looked for evidence of rate limiting from our chosen resolvers to ensure we were not generating too onerous a load, but saw no instances of failures related to rate limiting (e.g., many `SERVFAIL` responses or query timeouts).

2. Trufflehunter makes a DNS location request once per minute to determine the PoP towards which it is currently routing queries, for each public resolver. While the Ark nodes usually only change PoPs on the scale of days, they do go through occasional periods of "fluttering," where the PoP they are routed toward changes more frequently (on the order of minutes).

3.4.2 Finding unique TTL lines using cache probing

Recall from Section 3.3 that estimating cache occupancy for the four public resolvers we study requires counting the number of unique TTL lines observed from cache probes of a domain at each PoP. We now describe potential errors that can affect our estimate of unique TTL lines and our methods to mitigate these errors. Trufflehunter uses these methods in conjunction with the methods described in Section 3.3 to estimate cache occupancy of a domain at each PoP. Together, these methods yield a lower bounded estimate of the number of caches that contain a domain at a PoP within a TTL epoch.[5]

Error correction method for GPDNS, Quad9, and OpenDNS

Intuitively, counting the number of unique TTL lines (as defined in Section 3.3) will yield the number of caches that contain a domain. However, correctly identifying TTL lines using cache probing is challenging since DNS TTLs only have precision to one second. This lack of precision can introduce off-by-one errors when comparing TTLs in responses originating from the same cache. This situation may lead to significant overestimates of the number of TTL lines, and therefore also active users. Our goal, however, is to present *lower-bounds* on the number of

[5]TTL epoch is another term for the time period of the maximum TTL.

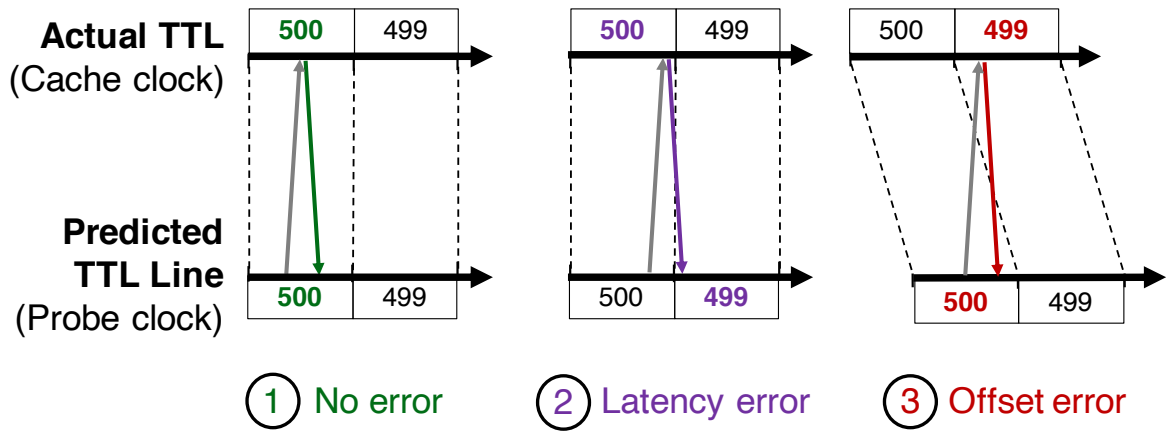

Figure 3.7. Errors matching cache probes to TTL lines

active users. We will now describe the nature of this problem in detail and describe the technique we developed to avoid overestimating the count of unique TTL lines.

To determine if a TTL returned by a cache probe lies on a particular TTL line, Trufflehunter needs two pieces of information: a sample of the actual TTL in the cache obtained by probing it, and a predicted TTL based on the probe clock's estimate of how much time has passed on the TTL line. Naively, one can determine if a TTL sample returned from the cache lies on the TTL line by checking if the predicted and actual TTLs are equal. However, TTLs in DNS responses only have precision to one second. Therefore, there can be sub-second measurement uncertainty. This uncertainty will lead to cases where Trufflehunter may overestimate the number of caches because TTLs do not lie on their predicted TTL line. Specifically, three cases can occur: the actual TTL matches the predicted TTL, the actual TTL is below the predicted TTL, or the actual TTL can be above the predicted TTL. The sources of uncertainty are as follows: the resolver's clock may not be synchronized with the probe clock, and there can be latency between when a resolver copies the TTL from its cache into a response, and when that response reaches the probing host. The effects of these sources of uncertainty are depicted in Figure 3.7.

First, consider the case where the resolver's clock is nearly synchronized with the cache probe's clock. In this case, TTL line prediction error will be due to the latency associated with cache probing.

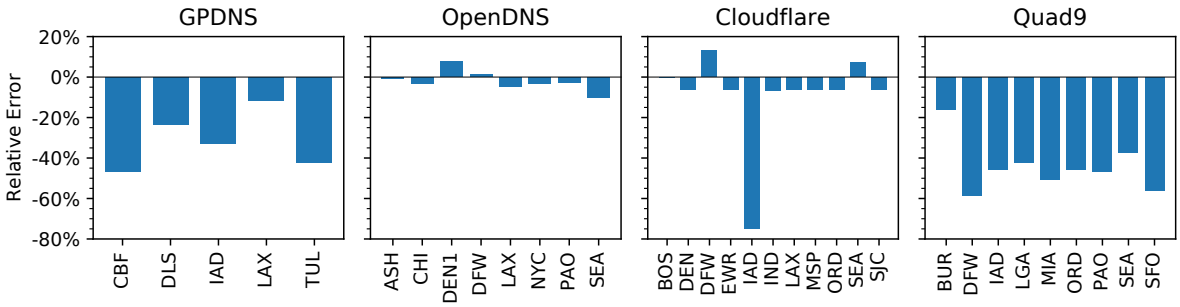

Figure 3.8. Cache estimation error

① There is no error when the TTL of the probe and cache stays the same between the time when a resolver generates its response and when the response arrives at the probing host.
② The TTL will be underestimated by one second when the probe TTL is decremented between the time when the resolver cache generates its response, and when it is received by the probing host.

Next, we consider the case where the clocks are not synchronized. ③ The actual TTL can be either overestimated, or underestimated, by one second. The error's direction depends on whether the probe clock is a fraction of a second ahead, or behind, the resolver's clock.

Trufflehunter uses the following heuristics to avoid overestimating the count of unique TTL lines due to TTL line prediction errors. If TTLs from cache probes lie on a single TTL line—with no probe TTLs falling on neighboring TTL lines (one second below or one second above)—we assume there are no errors. If we see a group of TTLs that lie on two neighboring TTL lines (one second apart), we assume there was an error, and we remove one line. If TTLs lie on a group of three or more neighboring TTL lines that are each one second apart, we remove the first and last lines in the group. Our rationale is that TTLs on the first TTL line may have been due to TTL overestimation, and the last line may have been due to TTL underestimation, but the TTLs on lines in the middle are likely composed of at least one correct measurement. This method can sometimes underestimate the count of TTL lines when lines that are one second above or below the predicted TTL line arise from other filled caches; however, this trade-off is consistent

with our goal of presenting lower-bounds on the number of active users. We have found that this technique is reasonably accurate on resolvers with no confounding factors, such as OpenDNS. In Section 3.5, we evaluate the effectiveness of this technique using controlled experiments. We found it allows us to estimate the number of caches (TTL lines) within approximately 10% of the true value on the resolver with the least confounding variables (OpenDNS).

Error correction method for Cloudflare

Because Cloudflare only has one shared, distributed cache per PoP, we do not apply the above error method to estimate how many caches each PoP contains. Instead, Cloudflare presents a different problem: its single externally visible cache is composed of many physical caches that all share the same record. The TTLs in this distributed cache drift away from the true TTL line over time, as the many individual caches that make up the single distributed cache share the record between themselves. As a result, TTLs from this cache get further away from the true TTL the longer the record is cached. Although it is easy to see visually that a group of DNS responses came from the same distributed cache, it is not trivial for an algorithm to do so: the drift allows a single cached entry to persist past the end of its TTL epoch. We use a peak-finding algorithm to find clusters of TTL lines that are near the true TTL line, essentially combining those TTL lines into one per TTL epoch.

3.5 Evaluation

In this section, we describe experiments to evaluate how accurately Trufflehunter estimates how many caches in a PoP contain a particular domain. In these controlled experiments, we use RIPE Atlas probes to mimic the behavior of many geographically distributed users querying for a domain from their local PoP. This allows us to quantify Trufflehunter's error in estimating how many *caches* in a PoP contain a domain within a TTL epoch, and therefore its error in estimating a *lower-bound* on the number of users that have queried for a domain within a TTL epoch. However, it does not reveal Trufflehunter's accuracy in estimating the number of

users over multiple TTL epochs (Section 3.6.1). The results of these experiments demonstrate that our estimates are consistent with Trufflehunter's goal of providing lower-bounded estimates of a domain's prevalence.

3.5.1 Simulating users with RIPE Atlas probes

Our goal in this experiment was to emulate users from multiple geographic locations requesting a domain from various public resolvers at diverse PoPs. Since Trufflehunter is deployed from the Ark infrastructure, we sought an orthogonal infrastructure that could provide this ability. RIPE Atlas probes are deployed in diverse locations; moreover, since this experiment did not have to be performed longitudinally, the restrictions with RIPE Atlas probes that prevented us from using the platform to run Trufflehunter long-term did not apply (such as low rate-limits towards destinations, low availability of RIPE Atlas credits etc.).

Choosing Atlas probes

We initially considered 956 RIPE Atlas probes located in the U.S. for this experiment, but realized that some showed evidence of having DNS requests hijacked by ISPs. If a request was hijacked, a request would arrive at the authoritative nameserver for the domain, but the cache that got filled as a result would not belong to the resolvers we were trying to measure and would therefore be invisible to Trufflehunter. We therefore designed an experiment to filter probes whose requests are hijacked.

We first asked each RIPE Atlas probe to request a subdomain, which contained its probe ID and its targeted resolver's name, from each of the four public resolvers. When we examined the IPs that requested these domains from our authoritative nameserver, we identified those that came from ASes that do not belong to the four public resolvers. We determined that the RIPE Atlas probes fell into three categories: some never had their requests hijacked (reliable probes), some always had their queries hijacked (unreliable probes), and some, to our surprise, seemed to have some of their queries hijacked but not others (suspicious probes). Further investigation

revealed that the suspicious probes were in small ASes that had multiple Internet providers. Our hypothesis is that one of these providers hijacks DNS queries and the other does not. Changes in routing could lead the probe's queries to switch between the hijacking provider and the non-hijacking provider. Another possibility might be that some ISPs hijack only a sample of DNS queries, not all of them.

We filtered 40 probes in this step, leaving 916 probes that could participate in the experiment.

3.5.2 Measuring cache fills from Ark nodes

We used the RIPE Atlas probes chosen from the previous step to repeatedly place a domain (whose authoritative nameserver is controlled by us) in the caches of the public resolvers. We simultaneously attempted to detect the presence of the domain with Trufflehunter. The RIPE Atlas probes placed the domain in cache by making recursive queries in bursts of ten minutes. These bursts were repeated at three hour intervals for a total of 48 hours. Note that this allows every individual cache to be filled up to sixteen times. Trufflehunter began searching for the domain several hours before the experiment began, when it was not yet expected to be in cache, and did not stop searching until many hours afterward. We did not detect the domain in any cache outside the duration of the experiment.

Figure 3.8 shows the accuracy of our estimate of the number of filled caches per PoP for each resolver. We calculate our error by calculating the percentage of the caches filled by Ripe probes that were *missed* by Trufflehunter. Missed caches are the caches filled by Ripe probes minus the caches observed by Trufflehunter. We count the caches filled by Ripe probes by enumerating the responses that have the maximum TTL for that domain. Both Trufflehunter nodes and the Ripe probes identify the PoP they are currently routed to by using the location queries from Section 3.4. Our error ranges from underestimating by approximately 10% on average on OpenDNS, to approximately 50% on average on Quad9. The difference in underestimation rate depends on the caching architecture of the resolver: some resolvers are easier to measure than

others. We note several interesting points from the results.

First, on OpenDNS, our method for eliminating error caused by measurements that have a granularity of one second appears to be reasonably successful. Recall that we eliminate the first and last TTL lines from each group, since we predict that they are likely to be composed only of measurements that are one second off from the true values (Section 3.4.2). We speculate that the remaining error is due to the fact that our error-removing technique is only a heuristic: there are a few lines that we remove that are not erroneous, and a few that we allow to remain that *are* erroneous.

We use the same technique on Quad9, but get very different results. Upon further investigation, it turns out that although Quad9 uses one DNS load-balancer called "dns-dist" for its frontend caches, they use two different software packages, Unbound and PowerDNS, for their backend resolvers. Unbound defaults to not answering queries with the Recursion Desired flag disabled; it returns a status of `REFUSED` [163]. Therefore, when a RIPE Atlas probe places its domain into the cache of a backend resolver that does not answer non-recursive queries, that record becomes invisible to Trufflehunter. We therefore underestimate the true number of filled caches at Quad9 PoPs. On most PoPs we underestimated by $\sim$50%, except for the BUR PoP, where our controlled users' queries appear to have coincidentally hit PowerDNS more than Unbound resolvers. Quad9's use of two backend resolver implementations is an interesting challenge, and one that limits the accuracy of our current technique.

On GPDNS, we appear to underestimate the number of filled caches by up to 45%, but this result may be due to the fact that the true number of filled caches is very hard to determine. Any request that missed in a frontend cache and hit in a backend cache presumably filled the frontend cache. Unfortunately, since it would not have caused a request to the authoritative nameserver, we cannot count the true number of filled caches with perfect certainty. We appear to have observed more caches than either RIPE Atlas's or Trufflehunter's queries would account for. However, since our estimate is consistently lower than the probable true value, this outcome is consistent with Trufflehunter's goal of providing a lower-bounded estimate of domain prevalence.

On most Cloudflare PoPs, Trufflehunter's error varies from 15% to −5%. Although each PoP's shared cache can only be filled once during a single TTL epoch, Trufflehunter cannot differentiate TTL epochs with perfect accuracy because the TTLs of the records drift over time (Section 3.4.2). IAD is the exception with significantly higher error (75%). For IAD, Trufflehunter only observed only the final four out of sixteen times the Atlas probes filled the cache. We suspect that there may have been a problem with the DNS location queries (Section 3.4) during this experiment. While both Trufflehunter and the Atlas probes recorded that they were using IAD, they may have been routed to different PoPs during the first twelve cache fills.

We also tested if any of a resolvers' PoPs appeared to use significantly different caching strategies compared to the resolver-specific strategies we identified in Section 3.3. We did not observe strong evidence of PoP-level inconsistencies.

In summary, Trufflehunter's cache enumerations underestimate by approximately 10-50% (excluding Cloudflare's IAD) depending on the resolver's cache architecture. Furthermore, since Trufflehunter consistently underestimates, our error does not prevent us from providing a *lower-bounded* estimate of cache occupancy. This allows Trufflehunter to fulfill its aim of enabling relative comparisons of rare domain popularity.

3.6 Case Studies

In this section, we apply our cache snooping technique to examine the use of three categories of abusive Internet phenomena: stalkerware, contract cheating services, and typo-squatting domains.

Our goal in this section is to provide lower-bounded estimates on the number of *users* of various domains using our estimates of the number of *caches* filled with these domains. During a single TTL epoch, local caches prevent a user from filling more than one resolver cache (Section 3.2), so every filled cache represents at minimum a single user. But when attempting

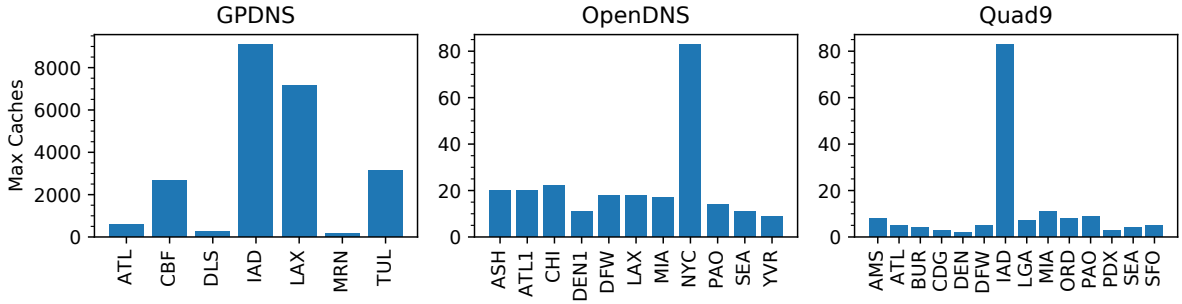

Figure 3.9. Maximum caches observed per PoP per resolver

to enumerate users, we would like to present estimates across *multiple* TTL epochs (such as the number of users in a day). This is more challenging—it is hard to distinguish between one user making multiple queries in distinct epochs and multiple users making individual queries. If the rate at which users make requests can be determined, it is possible to deduce the number of individual users from the observed filled caches [179]. Unfortunately, the intervals at which a user visits a website are not usually deterministic. Without a vantage point that can grant insight into user behavior, estimating the number of unique visitors to a site is difficult. It is reasonable to assume that a user will not make multiple requests for a domain within a single TTL epoch, because of the caching behavior of operating systems and browsers (Section 3.2). But the number of unique website visitors cannot be estimated at time scales larger than a TTL epoch. We therefore present our estimates of website traffic in the form of web requests per day, summed over all resolvers and PoPs.

However, domains that are associated with *applications* instead of websites are often accessed automatically, without user interaction, at regular time intervals. Such regularity provides the opportunity to estimate the number of unique application users with better accuracy than we can estimate unique website visitors. Unfortunately, this insight still does not allow us to distinguish between unique users in multiple TTL epochs. As a result, we use the maximum number of users ever observed during a single TTL epoch as a conservative lower bound estimate of the number of users of a given application. This "maximum users per epoch" metric sums

users observed across all PoPs and resolvers during that epoch. We assume that an individual user is unlikely to make DNS requests to either multiple PoPs or multiple resolvers during one TTL epoch.

3.6.1 Limits on observed users

Recall that our lower bound estimates of the number of users of a domain depend upon the number of caches at each PoP that can potentially contain a domain. If there is only a single shared cache at a PoP (as in Cloudflare's case, for example), we can only estimate at most one user per TTL epoch at that PoP. However, as the number of available caches increases, so does the opportunity to observe more users filling caches.

Trufflehunter can observe at most one user or web request per cache per TTL epoch. We can quantify our error in estimating how many caches have been filled over time (Section 3.4.2), but we cannot estimate how many more users have accessed a cache after it has been filled by the first user, until the cache expires and is refilled. This is the reason that Trufflehunter can only provide a lower-bounded estimate of the users of an application or visitors to a domain.

We used the data collected by Trufflehunter between March 6, 2020, and May 30, 2020 (per the methodology described in Section 3.4.1) for the domains in our study to estimate the number of individual caches that each PoP contains.[6] We counted the maximum caches that Trufflehunter saw during this period filled with any of the domains we studied during any single TTL epoch. Figure 3.9 shows these results. We do not include Cloudflare in Figure 3.9 since the maximum number of caches in any PoP is always one due to Cloudflare's use of a single shared cache per PoP. We note that GPDNS appears to have thousands of caches, consistent with our model of GPDNS in Section 3.3. This allows us to observe thousands of users per TTL epoch at each GPDNS PoP. On the other end of the scale, Cloudflare only has one cache per PoP. We do note, however, that although Cloudflare has the fewest caches per PoP, it has the most PoPs

[6] We elided a handful of days from our data because a small number of Ark nodes constructed some queries incorrectly on these scattered days, and may have poisoned the resolvers' caches by placing domains in them.

in the U.S. out of the resolvers we studied (46). This allows us to observe a non-trivial number of Cloudflare users across the U.S., and Cloudflare users do contribute to our total estimates of users (Figure 3.10). We also observe that across resolvers, larger PoPs like IAD and NYC have more caches, as might be expected.

3.6.2 Stalkerware

We first apply Trufflehunter to estimate the prevalence of stalkerware. The term "stalkerware" covers a wide range of software used in the context of intimate partner violence (IPV). It is installed by the abuser onto the target's device, usually a cell phone. Apps vary widely in their range of capabilities, which can include tracking location, recording messages sent by text or other messaging apps, recording audio of phone calls and ambient sound, spoofing texts to the target, and more. The abuser then accesses the target's information by visiting an online dashboard, which is updated regularly by the app. Stalkerware broadly falls into two categories: "dual-use" apps, designed for a benign purpose and repurposed as spyware, and "overt" apps, which hide their presence on the target device and often have more dangerous capabilities than dual-use apps. Some are explicitly marketed for catching an unfaithful partner or spouse, although this messaging recently appears to have become more subtle or disappeared entirely [146, 147]. Since even overt applications are now advertised for legal or legitimate uses such as parental control, it must be noted that we have no way to tell whether Trufflehunter is observing stalkerware used in the context of IPV, or stalkerware installed for other reasons.

In the context of IPV, previous research has taken a clinical approach to studying stalkerware, and has uncovered little evidence of the use of overt applications. Most digital surveillance previously uncovered appears to be facilitated by either dual-use apps or misconfigured settings [61, 105].

However, a clinical approach has limited scalability: the researchers were able to speak with fewer than fifty IPV survivors. In contrast with clinical studies, a quick Google search reveals dozens of overt stalkerware applications, as well as anecdotes and articles describing

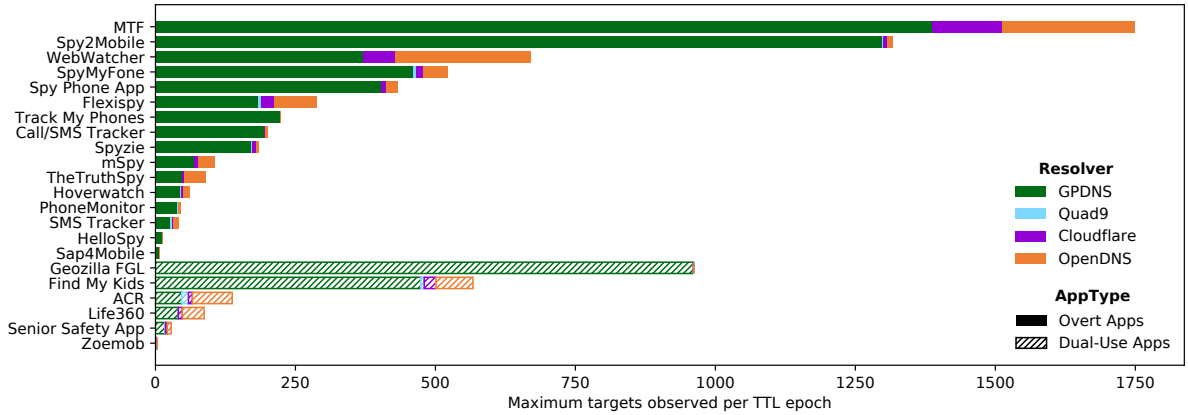

Figure 3.10. Stalkerware targets visible per TTL epoch

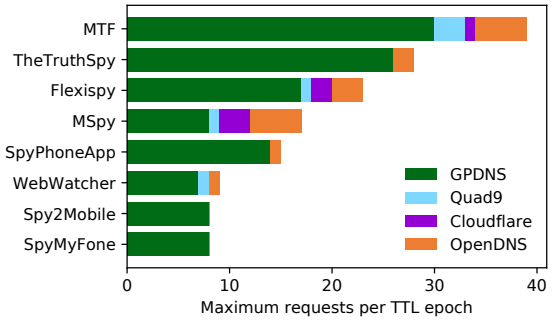

Figure 3.11. Web requests per TTL epoch for stalkerware dashboards. Note that not all stalkerware apps have dashboards.

what targets of the overt applications' capabilities have gone through. The contrast begs the question: how many people have overt stalkerware installed on their phones? Even if overt stalkerware is not the most common vector for digital stalking, it is a dangerous phenomenon worth understanding in more detail.

Our application user measurement technique is uniquely suited to measuring stalkerware because it leverages the knowledge that some apps make requests automatically at well-defined intervals. Stalkerware apps often exhibit this behavior, without requiring any user interaction. In fact, since they are installed on devices whose owners are unaware of their presence, this behavior is a necessity.

Profiling Stalkerware Applications

To model stalkerware request behavior, we examined the network traces of approximately 60 stalkerware apps on an Android Pixel phone. We found these stalkerware apps using a combination of searching the Google Play store and Google search results: previous work [61] indicates that spyware is surprisingly easy to find this way. We discarded around 40 apps that did not function correctly or were not usable as spyware. The remainder are a combination of dual-use and overt apps.

We then installed each application one at a time and recorded its network trace for a few hours. During each recording, we occasionally sent messages, made calls, installed apps, and simulated other behavior that the stalkerware claimed to track. We identified most of the domains that each app requested in this manner. We then installed all of the applications at once and recorded the phone's network activity during specific activities, such as sending a text, making a phone call, or rebooting the phone. The goal was to determine if any apps reacted to the target's activity on the phone by making network requests, or if the apps simply send information at regular intervals regardless of target behavior. We found that most apps did not appear to respond to target actions, with the exception of reboots: most apps made network requests directly after the phone was restarted. Finally, we recorded the phone's network traffic for a total of eighteen days, while attempting to use the phone like a normal device. Using this data, we determined which apps make requests at regular intervals. For the apps that do not, we make the most conservative assumption: that they make requests at most once per TTL epoch.

We make the simplifying assumption that targets have one device with stalkerware installed. We also assume that an individual device will not access more than one PoP or more than one resolver during a single TTL epoch. Therefore, for each domain used by an app, we first calculate the sum of all the caches we observed to be filled with that domain across all PoPs and resolvers for every TTL epoch. We then divide this sum of filled caches for every TTL epoch by the app request rate we calculated with our network traces. This yields the number of targets

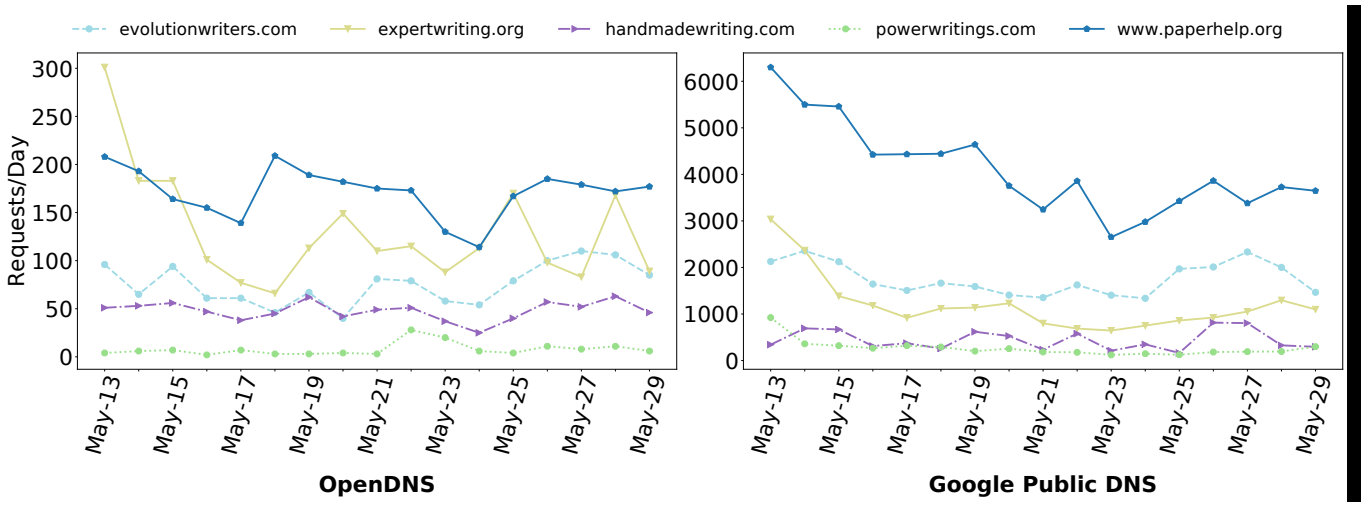

Figure 3.12. Web requests per day of contract cheating services.

visible during each epoch. Finally, we calculate the maximum number of targets ever visible during one TTL epoch as a lower-bounded estimate of how many targets of stalkerware exist in the U.S.

Estimates

Figure 3.10 shows the maximum targets ever observed in a single epoch for 22 stalkerware apps between March 6, 2020, and May 29, 2020. Overt apps are shown as solid colors and dual-use apps have hatches. Each app is broken down by the resolver at which the targets were observed. Most targets are observed in GPDNS's caches, since GPDNS has the most caches per PoP. Interestingly, during the particular TTL epoch when the maximum users were observed, some apps, such as WebWatcher, SpyPhoneApp, and others, were not present in Quad9's caches. We speculate that this could be due to Quad9's use of Unbound resolver software, which prevents Trufflehunter from observing some filled caches.

We estimate that a minimum of 5,758 people are targeted by overt stalkerware in the U.S. today. The most popular overt app, Mobile Tracker Free, accounts for a third of observed targets (1,750). In contrast to most subscription-based overt stalkerware, Mobile Tracker Free and Spy2Mobile (the second most frequently observed app) are both free, which likely accounts

for their high popularity. Additionally, Spy2Mobile is one of only two overt apps we studied that is available on the Google Play store (the other is Call/SMS Tracker). All other overt apps must be downloaded from third-party websites.

We also used Trufflehunter to observe web requests for the dashboard websites that attackers use to view targets' information. Because web requests made by attackers do not exhibit periodic behavior, Figure 3.11 shows the maximum number of web requests ever visible during one epoch. We note that the popularity of app dashboards does not always correspond to the prevalence of the app, likely due to the features the app provides. For example, abusers might check Spy2Mobile's dashboard less frequently than MTF's because Spy2Mobile primarily provides location data, while MTF also records messages, phone calls, and more.

3.6.3 Contract cheating services

We next use Trufflehunter to examine users visiting "contract cheating" domains. Contract cheating services offer to complete students' homework assignments, projects, and in some cases entire classes for a fee. It is an increasingly popular method of cheating since it does not rely on plagiarism and is therefore more difficult to detect [66, 214]. The specific services provided include essay-writing services, "agency sites" that use auction models to match students to contractors who can complete assignments, copy-edit services, and more [136]. Since cheating is by its nature something that students are reluctant to admit to, it is difficult to measure using indirect means such as surveys.

We identified a set of ten popular contract cheating websites based upon search results and online discussions and recommendations. From May 3–29, 2020, we used Trufflehunter to track activity to these sites. Figure 3.12 shows the daily sum of web requests observed over time across the two resolvers with the most activity. Interestingly, we note a decrease in some services towards the end of May, perhaps because schools and universities that use semester systems were transitioning to summer break.

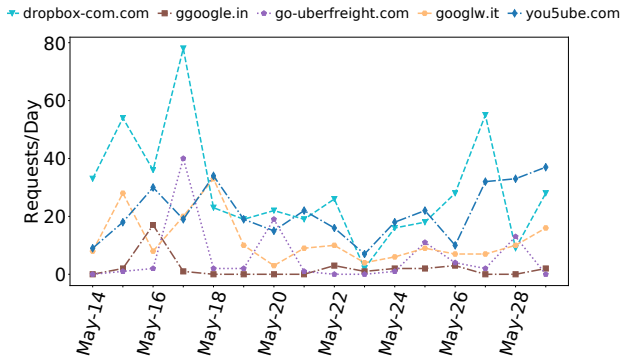

Figure 3.13. Web requests per day for typo-squatting domains on GPDNS

3.6.4 Typo squatting domains

Finally, we use Trufflehunter to estimate users visiting typo-squatting domains. Typo-squatting is the abusive practice of registering a domain similar to a popular domain so that users will be tricked into mistyping or clicking on it. Some forms of typo-squatting simply seek to show unwanted advertisements to the user, which is irritating but not generally harmful. Others distribute malware, or imitate children's websites and redirect to adult websites, or trick users into entering credentials which are then stolen [216].

To find typo-squatting domains that were likely to be active, we first tried to resolve the typo squatting domains listed in Tian et al.'s study [205]. We attempted to remove domains that did not appear malicious at the time of the study, such as paypal-cash.com, which is now an apparently legitimate domain owned by PayPal. Figure 3.13 shows the snooping results for the five most-accessed typo-squatting domains from May 3–29, 2020, on GPDNS, the resolver that showed the most activity. Given that these domains were in use two years before our study, it would not be surprising if all were on blacklists. We found that some domains were still surprisingly active, with dozens of resolutions per day.

We also looked for several domains used by hack-for-hire services in spear phishing attempts [154]. Of all of these, 18 still resolved to an IP address as of May 2020, and only one made an infrequent appearance in any resolver's cache.

3.7 Related Work

Various aspects of recursive DNS resolver behavior, including caching [122, 121, 157], client proximity [150, 166, 62, 34], and vulnerabilities [197, 190, 106] have been studied. We focus our attention on studies that investigated caching behavior in public DNS resolvers and prior applications of cache snooping.

With public DNS recursive resolvers increasing in popularity [55], their caching and load-balancing behavior has received attention from the community. Callejo et. al. observed that public DNS resolvers including GPDNS, OpenDNS, Level3, and Cloudflare are responsible for 13% of the DNS requests in their online-advertising-based measurement campaign [55]. Public DNS resolvers use anycast [158] and can be present at multiple PoPs [73]. Some studies have observed that public resolvers have multiple caches for load-balancing [129, 192], which can be fragmented [30, 217, 158]. While these studies have investigated different aspects of the caching behavior of public resolvers, ours is the first to enable DNS cache snooping on them.

Several studies have used DNS cache snooping to measure various domains. Wills et al. used this technique in 2003 to measure the popularity of various web domains [226] and in 2008, two other studies used DNS cache snooping for similar purposes [179, 32]. Rajab et al. measured the relative footprints of various botnet domains [28, 180], and Kührer et al. used cache snooping to analyze which "open" resolvers found in an Internet-wide scan were actively providing service to clients [126]. All these studies assume that the resolvers they are probing have a single cache; our work has demonstrated that this assumption is no longer valid, especially for public DNS resolvers. Since these efforts did not focus upon potentially sensitive domains, they were able to probe the caches of arbitrary "open" resolvers. However, recent work has shown that millions of open resolvers are misconfigured residential devices that are unintentionally open [190, 189, 71, 33, 126], and are therefore not suitable for use in our study.

More recently, Farnan et al. used cache snooping on recursive resolvers belonging to VPN providers to analyze which domains are accessed through VPNs [90]. They target recursive

resolvers belonging to VPN providers, which do not appear to have the complex caching architectures we observed in public resolvers.

Our work is the first to successfully demonstrate that public DNS resolvers can yield meaningful estimates of active users in a privacy-conscious way due to their underlying caching properties.

3.8 Ethics

Since DNS cache snooping can reveal if a domain was recently accessed by the users of a DNS resolver, some ethical questions arise that we address below.

First, if a DNS resolver is used only by a few users, cache snooping may identify with fine granularity which domains these users accessed, impinging upon their privacy [99]. We avoid this issue by targeting our measurements only at large, public resolvers with thousands of users. Doing so allows us to measure how often anonymous users access rare domains and yet learn little that could aid in deanonymizing individual users. We refrain from probing caches of other "open" resolvers, since these are often misconfigured residential devices that may be serving only a few users [190, 189].

Second, domain names could contain user identifiers that potentially enable identifying a user's activity in a resolver. For example, a service may embed usernames into a unique service subdomain that may be periodically requested from a user's device. While a potential side channel for a resolver of any size, in our work we do not probe any domain that contains user-specific identifiers, and no individual user's information is exposed by our measurements.

Third, some users may be motivated to use large public resolvers because of the increased resilience to cache snooping they seemingly provide. Since the technique we describe in this paper identifies a side channel using the combination of rare applications and the caching architecture of large resolvers, we will be notifying the public resolvers of our findings.

Finally, the applications that we use in our case studies are by their nature sensitive. Since

we are not able to identify individual users, though, we cannot associate the use of sensitive applications with any particular user. At most we can identify that these applications are in use at the coarse granularity of a large geographic region served by a PoP.

3.9 Summary

This chapter introduces Trufflehunter, a privacy-preserving DNS cache snooping that models the caching architecture of public resolvers to provide lower-bounded estimates of cache occupancy. Applied to four large public DNS resolvers, Trufflehunter achieves a 10% to -50% error. Trufflehunter may be applicable to more distributed public DNS resolvers than the four we studied in the paper. Indeed, we observed the same caching strategy in Quad9 and OpenDNS. Trufflehunter provides lower-bounded estimates of domain usage, therefore it is best suited for relative comparisons of rare domain popularity, rather than for estimating an absolute number of users per domain. To demonstrate this capability, we showed how to estimate the prevalence of rare and sensitive applications on the Internet, which are otherwise difficult to measure from a third-party perspective.

Chapter 3, in part, is a reprint of the material as it appears in *Proceedings of the Internet Measurement Conference 2020*. Audrey Randall, Enze "Alex" Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Home is Where the Hijacking is: Understanding DNS Interception by Residential Routers

While performing the experiments in the last chapter, we discovered that some of our DNS queries were returning misformatted responses. We eventually determined that the queries were getting transparently intercepted and redirected to a different resolver than the one we had specified. This phenomenon underscores two key takeaways. First, no matter what the documentation of a naming system says, there is always a possibility that the system works differently in practice than it does in theory. Naming systems can have unexpected components and participants. If a defender attempts to determine which component of a system is most impactful to deny to an adversary, they must be certain which ones are actually getting used.

Second, the presence of unexpected actors within a system presents an opportunity. When multiple entities in a naming system share a role (for example, multiple intermediaries resolving queries), the details of their implementations may differ. A researcher can exploit these subtle implementation differences, such as the format of query responses, to create "fingerprints" for each entity that plays a role in a naming system. Identifying the participants in a naming system can inform interventions by accurately enumerating the resources an adversary requires. Thus, the goal of this next study is to determine if the presence of unexpected actors in naming systems can be observed by their users, by determining the location in the network where DNS redirection

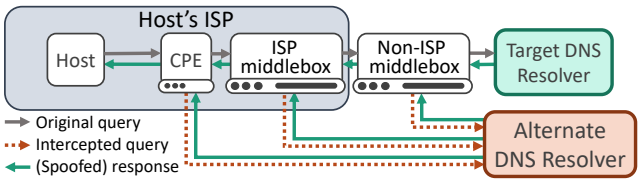

Figure 4.1. Locations where interception can occur.

occurs.

4.1 Overview

In principle, devices are free to direct their DNS queries to the recursive resolver of their choosing. Indeed, it is this freedom that has enabled the growth of public resolvers such as those offered by Google, Cloudflare, and others. However, a key underlying assumption is that DNS queries are faithfully forwarded as they are addressed. Unfortunately, this is not always so.

DNS queries sent by a user's device can be *intercepted* en route to a target resolver and forwarded to an alternate resolver. Furthermore, this interception can be *transparent*, where the interceptor spoofs responses so they appear to have been sent by the intended resolver. Transparent interception is difficult to detect because the alternate resolver does *not* have to modify the response. Even if the reason for the interception is benign — such as to prevent malware from evading DNS filtering — the interception of requests and misrepresentation of responses raise serious ethical concerns [211, 68] and can also interfere with the correct operation of protocols such as DNSSEC [142, 68].

While prior work has identified the broad prevalence of transparent interception [142, 120, 222], there are no established techniques for establishing *where* the interception is implemented. Indeed, there are a range of different points in the network where such interception might take place.

DNS redirection, another form of DNS manipulation, has also been found to occur in several parts of the network. DNS redirection occurs when a DNS resolver returns an altered response for specific queries and may occur with or without DNS interception. DNS redirection

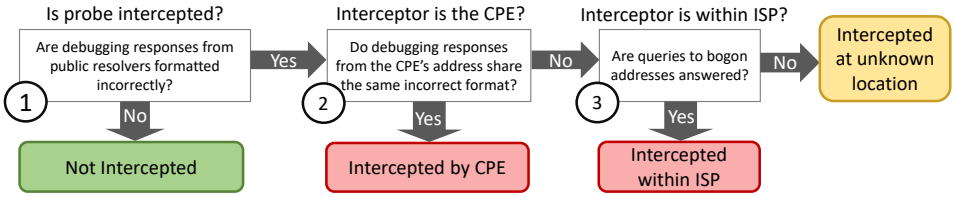

Figure 4.2. Three-part technique to determine if and where a DNS query is being intercepted.

has been discovered *in Customer Premises Equipment (CPE)* to block resolution of specific domain names [83], *in ISPs* to replace NXDOMAIN responses with advertisements [218, 135] or enhance security and performance [39], and *outside of ISPs* to implement country-level censorship [41, 81, 43, 126]. Transparent interception has been far less extensively studied, although we are aware of anecdotal reports suggesting that, in some cases, the DNS forwarder in CPE [104] may be implicated [92, 67, 64].

In this chapter, we develop a technique that can isolate *where* in the network transparent interception is occurring. In particular, we make the following contributions:

- **Identifying Where Transparent Interception Occurs**. We demonstrate how targeted use of standard DNS *debugging queries*—such as `id.server` and `version.bind` [227]—can identify not only the presence of transparent interception but to systematically infer if the source of that interception is the CPE or middle-boxes in the ISP (Figure 4.1). Our technique can be implemented on any device that can make DNS queries, without requiring root access or external measurement tools such as an authoritative nameserver.

- **Pilot Study of Transparent Interception on RIPE Atlas**. We demonstrate our technique on RIPE Atlas infrastructure as a pilot study. We identify over 200 occurrences of such transparent interception. Of these, we further show that CPE-based transparent interception constitutes a significant fraction.

- **Case Study: CPE-based Transparent Interception**. We highlight a particular case that more precisely explains the nature of the DNS interception mechanism. Specifically, we illustrate how the common XB6/XB7 home router (used by many ISPs) use Destination

Network Address Translation (DNAT) to transparently intercept queries and forward them to the ISP's resolver.

We believe that techniques such as these, that discover how and where DNS manipulation takes place, are especially important in light of the ongoing public controversy concerning the value and implementation of privacy-enhancing DNS transport [107, 110].

4.2 Background and Terminology

During normal DNS resolution, a user device sends DNS request(s) to a *target resolver* (the user's ISP's resolver, a public resolver such as Google, Cloudflare, etc.). The target resolver recursively resolves the request and returns the response to the user device. Adopting terminology from Liu *et al.* [142], we refer to instances where DNS queries to a target resolver are intercepted and forwarded to an *alternate resolver* as **DNS interception**. DNS responses arriving at the user device from these alternate resolvers arrive with the source address spoofed to be that of the target resolver [142]; if not, the response would be rejected by the user device. When DNS responses are spoofed in this manner, the interception is *transparent* to the user device, and is difficult to detect. We refer to transparent interception as simply "interception" in the rest of the paper.

DNS redirection is a related but different form of DNS manipulation, where DNS responses — often NXDOMAIN responses [135, 65] — are altered, resulting in users getting redirected towards a different resource from the one in the unaltered response. It is often performed by the *target resolver*, rather than by forwarding a query to an alternate resolver. The insight that the response received by the user is *different* from the correct response during DNS redirection has been used to study this phenomenon in detail [218, 135, 39].

DNS interception, in contrast, has received less attention, and is the focus of our paper. In 2018, Liu *et al.* measured the prevalence of DNS interception [142], but did not measure *where* in the network their queries were intercepted. We present a technique that can both determine

Table 4.1. Location queries and examples of expected responses from each resolver.

Public Resolver	Type	Location Query	Example Responses
Cloudflare DNS	CHAOS TXT	`id.server`	`IAD`
Google DNS	TXT	`o-o.myaddr.l.google.com`	`172.253.226.35`
Quad9	CHAOS TXT	`id.server`	`res100.iad.rrdns.pch.net`
OpenDNS	TXT	`debug.opendns.com`	`server m84.iad`

whether a DNS query is intercepted, as well as if it was intercepted by the client's own CPE or ISP.

4.3 Methodology

In this section, we present our methodology for detecting where DNS interception occurs. Figure 4.2 illustrates the three steps of the technique and the information used to make determinations at each step. We next describe each step in more detail, and present a concrete example of the technique in practice.

4.3.1 Identifying query interception

We show that it is sufficient to use a few select *location queries* — for which it is difficult to spoof the correct response — to detect query interception. Public anycast resolvers implement these queries to aid debugging, by revealing the location of the specific server that answers the query [88]. Our technique issues location queries to four public resolvers (on both primary and secondary IP addresses) and tests for "non-standard" responses to discover query interception. This technique is similar to the one used by Jones *et al.* to detect DNS root manipulation using `hostname.bind` queries [120]. Moreover, since each public resolver has both IPv4 and IPv6 addresses, we can detect interception in both protocols.

Each of the four public resolvers that we study implements its own version of a location query. Table 4.1 lists the queries and an example expected response. Each resolver uses a different format for its responses, and these formats are consistent around the world. We determined

"standard" responses for each resolver by making requests from a network that we knew did not experience interception, and later confirmed that these responses were the expected ones in conversations with public resolver operators. When testing for interception, we compare responses to location queries issued from the device under study against these standard responses; when a response does not match the standard response, we conclude that the query has been intercepted.

We note that if a query is dropped entirely, it will appear to the client as a timeout. While timeouts can potentially reveal interesting behavior, such as censorship, for the purposes of our study we conservatively assume that timeouts are not due to transparent interception. We also note that prior work has observed query *replication*, where two responses are sent to the client: one from the intended recipient, and one from the interceptor's chosen resolver [142]. However, the interceptor's response nearly always arrives first and is accepted by the client, so interception and replication are indistinguishable for our purposes.

4.3.2 Identifying query interception by the CPE

After we determine that interception is occurring using location queries, we then find where the query is first diverted away from its intended destination. We begin by using a novel technique to determine if the client's CPE is responsible for the interception.

First, we issue a `version.bind` query to the CPE's own public IP address. By usual IP routing rules, this query cannot travel beyond the CPE because the CPE is its destination. However, if the CPE is the interceptor, it will switch roles at this point: rather than acting as a packet forwarder following IP rules, the CPE will take on the role of a DNS forwarder instead. This role switch occurs because the most common method of implementing interception is DNAT. DNAT rewrites all query destinations to be the CPE's own private IP address, so that the CPE's DNS forwarder (e.g., Dnsmasq) can send them to its own pre-configured resolver. If the CPE's DNS forwarder supports the `version.bind` request, it will not forward the query any further, and will directly return a response.

However, this result alone is insufficient to demonstrate that the CPE is the interceptor, because there is another circumstance that could allow the CPE to forward the query: if the CPE's port 53 is open, it will act as a DNS forwarder even if it is not an interceptor. To distinguish between these cases, we next issue `version.bind` queries to each of the public resolvers we study. While only one resolver (Quad9) answers `version.bind`, it is immaterial — if the CPE is the interceptor, it will answer the query instead, and produce the same response as the query sent to the CPE's public IP address. We then compare the response strings from the query to the CPE with the responses from the queries to the public resolvers. If they are identical, we may conclude that the CPE is using DNAT to intercept queries to that resolver. (For more details on why we use `version.bind` for this technique, please see Chapter 4.4.)

4.3.3 Query interception by the ISP

If the interception is not being performed by the CPE, we next check whether it is occurring within the ISP. We can identify interception within the ISP by using another novel technique: making DNS requests to bogon IP addresses ("bogon queries"). Bogon IPs are unroutable, so bogon queries should not be able to leave the AS in which they originated. We chose one IPv4 and one IPv6 bogon address, confirmed that queries to these IPs were not routable, and directed queries for a generic domain we control to both IP addresses. If we received a response, we concluded that the request must have been intercepted before it could leave the AS. If we did not receive a response, two possibilities exist: either the interceptor was outside the AS, or the interceptor discards queries to unroutable addresses. Thus, if we received no response, we cannot determine where the interceptor was located. We found that most interception in most countries occurs before the query exits the AS (Figure 4.4).

4.3.4 Example of technique in practice

Tables 4.2 and 4.3 illustrate the technique using three RIPE Atlas probes and their responses. The first step tests if any of the probes are being intercepted. Table 4.2 shows the

Table 4.2. Example responses to IPv4 location queries.

ProbeID	Cloudflare DNS	Google DNS
1053	SFO	172.253.211.15
11992	*NOTIMP*	62.183.62.69
21823	routing.v2.pw	185.194.112.32

Table 4.3. Example responses to IPv4 `version.bind` queries.

ProbeID	Cloudflare DNS	Google DNS	CPE Public IP
1053	-	-	-
11992	*NOTIMP*	*NOTIMP*	*NXDOMAIN*
21823	unbound 1.9.0	unbound 1.9.0	unbound 1.9.0

location queries to the IPv4 addresses of Cloudflare DNS and Google DNS. Probe 1053 receives an expected response, hence the queries are not intercepted: Cloudflare returns a three letter IATA airport code, and the Google responses map to Google IP addresses. On the other hand, probes 11992 and 21823 have non-standard responses, so their queries were intercepted.

Next, we issue `version.bind` queries for the two probes that were intercepted. Table 4.3 shows the responses. For probe 21823 the responses from Cloudflare DNS, Google DNS, and the CPE's public IP address are all the same, which indicates that the CPE is the interceptor (Section 4.3.2). For probe 11992 the responses are a mix of `NOTIMP` and `NXDOMAIN` responses, so the CPE was not the interceptor in this case.

Finally, we determine if probe 11992 was intercepted within the ISP by issuing a query to a bogon IP address. Because bogon IP addresses are not routable, the ISP should drop the queries. However, if the responses are valid and match the responses purportedly from the public resolvers in Table 4.3, then we can conclude that the interception happened within the ISP. If not, we cannot draw a conclusion about where the interceptor is located within the network. In the case of 11992, we received a NOTIMP response to the bogon query, and thus concluded that 11992's interceptor was within its ISP.

4.4 Why version.bind is necessary to detect CPE interception

We identify cases where the CPE is the interceptor by sending a specific CHAOS TXT query for `version.bind` to the public address of the CPE. Under ordinary routing rules, the CPE should not forward this packet to any other destinations, so if we receive a response to this query, we know the CPE has not obeyed usual routing rules and might be the interceptor. A reader might ask why it is necessary to send `version.bind`, which some resolvers are not configured to answer, rather than any DNS request for an ordinary A record: if we receive an answer for an ordinary A record request, does this indicate that the CPE is the interceptor?

Our reasoning is that it does not, and our logic is as follows. When a DNS request for an A record is sent to the CPE's public IP address, if the CPE's port 53 is open, even a non-intercepting CPE will return a response. This behavior is even true for `version.bind` queries. Therefore, the result of a single query to the CPE's public IP address is not sufficient to determine if the CPE is the interceptor. Our method relies on *comparing* the `version.bind` query sent to the CPE's public IP address with the `version.bind` query sent to the public resolver. The answer to a `version.bind` query is a string that is much more unique than an ordinary DNS response, and this property is necessary for determining if the CPE is the interceptor.

Consider the following scenario if an ordinary DNS query were used to determine the interceptor, leading to an incorrect conclusion. Let us assume the CPE is not the interceptor, but it does have port 53 open. If we were to send a query for `example.com` to the CPE's public IP address, the CPE would forward that query to its DNS resolver (for example, Comcast DNS) because its port 53 is open. We would receive the IP address of `example.com`, for example, "1.2.3.4." Next, we send a query for `example.com` to a public DNS resolver like Google DNS. The CPE is not the interceptor, so it sends the query towards Google DNS as intended. The query is intercepted further along the path, but no matter which resolver eventually answers it, the response is "1.2.3.4."

Now consider the case where the CPE *is* the interceptor. Both queries for `example.com` would be forwarded to the CPE's resolver, and both answers would come back as "1.2.3.4." We cannot tell whether the CPE was the interceptor because all answers to our queries are identical. The advantage of using `version.bind` is that it returns a more unique string. If the CPE is not the interceptor, but does have port 53 open, the query to the CPE's public IP address will return its own answer to `version.bind` (e.g., "Dnsmasq 2.7."). The query to a public resolver such as Google DNS will arrive at some non-CPE resolver further along the path, and will return that resolver's answer to `version.bind` (e.g., "PowerDNS"). The CPE's response to a `version.bind` query is unlikely to be identical to the intercepting resolver's response. But if the CPE is the interceptor, both `version.bind` queries will be handled by the CPE's resolver, and they will return identical answers. We can therefore determine with high confidence when the CPE is the interceptor.

4.5 Ethical Considerations

Our work does not raise ethical concerns as we issue standard DNS queries towards major public DNS resolvers from a platform with volunteer-consent for such measurements.

4.6 Pilot Study on RIPE Atlas

We use the RIPE Atlas platform to perform a pilot study that confirms our technique works in the wild. With RIPE Atlas we can launch DNS measurements from roughly 10,000 probes around the world [198]. However, RIPE Atlas is not representative of the Internet as a whole: it has significantly more probes in Europe and North America than anywhere else, and also has a "geek bias" due to its volunteer-driven deployment. These biases should be taken into account before generalizing our findings on DNS interception to ISPs around the world. However, we emphasize that our technique is broadly transferable. With a handful of DNS queries, we can determine not only if queries are being intercepted, but also where the interception is occurring.

Table 4.4. Number of intercepted probes per public resolver.

	Resolver IPv4		Resolver IPv6	
	Intercepted	**Total**	**Intercepted**	**Total**
Cloudflare DNS	165	9619	11	3730
Google DNS	160	9655	15	3726
Quad9	156	9616	11	3732
OpenDNS	156	9666	11	3727
All Intercepted	108	9537	0	3691

We therefore believe our technique can be easily deployed on other measurement platforms as well.

4.6.1 Which probes experience interception?

As described in Section 4.3.1, the first step of our technique identifies which probes experience interception. We do so by sending *location queries* from every RIPE Atlas probe worldwide that would respond to our measurement requests. Over 9,600 probes responded to at least one experiment (Table 4.4). We identified 220 RIPE Atlas probes that experience the type of interception we were looking for, which we now break down by their location and behavior.

Which public resolvers were subject to interception?

We test interception with four public resolvers: Google DNS, Cloudflare DNS, Quad9, and OpenDNS. Table 4.4 shows that the majority of intercepted nodes experienced interception for all four public resolvers. If fewer than four resolvers experienced interception, the most common pattern was either that only one resolver was intercepted, or only one resolver was *allowed*. In the former case, Google DNS and Cloudflare DNS were intercepted more often than Quad9 and OpenDNS, perhaps because of their popularity and market share. In the latter case, we hypothesize that the interceptor is deliberately using a single public resolver, perhaps for malware filtering purposes or ease of implementation.

We note that most interceptors that act on IPv4 queries for a public resolver *do not*

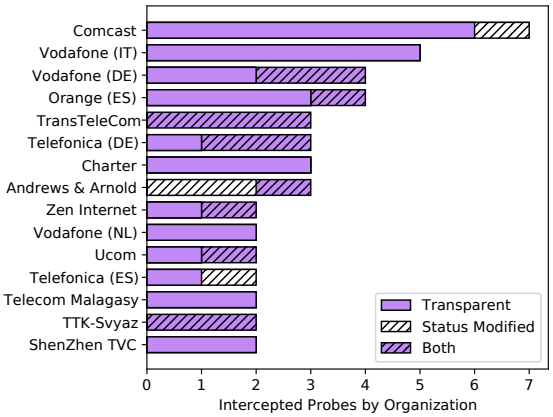

Figure 4.3. Intercepted probes per top 15 organizations.

intercept queries for that resolver's IPv6 addresses. Table 4.4 shows that only a handful of probes experience both IPv4 and IPv6 interception. Because IPv6 interception is infrequent, we consider IPv4 and IPv6 jointly for all subsequent analyses.

Is interception transparent?

If an interceptor intends to be transparent, we assume it will correctly resolve most DNS queries. If it did not, it would be obvious to the client that something was wrong. To test this hypothesis, we sent a request for `whoami.akamai.com` [134] to all four public resolvers from each intercepted probe. We do not expect this domain to be blocklisted.

The answers to this query let us confirm (a) that interception is indeed occurring (if the returned IP address is *not* one of the egress addresses of the target resolver) and (b) that the interception is transparent (if we do not receive an error in the response).

Figure 4.3 categorizes the responses. The "Transparent" bar indicates that queries to all intercepted resolvers were unchanged, "Status Modified" indicates that queries to all intercepted resolvers returned DNS error statuses, and "Both" indicates that requests to some resolvers were transparently intercepted while requests to others received modified status codes. The majority of queries across countries and ISPs return a valid response, which indicates that even intercepted queries *are* resolved correctly—just not by the targeted public resolver. However, some queries

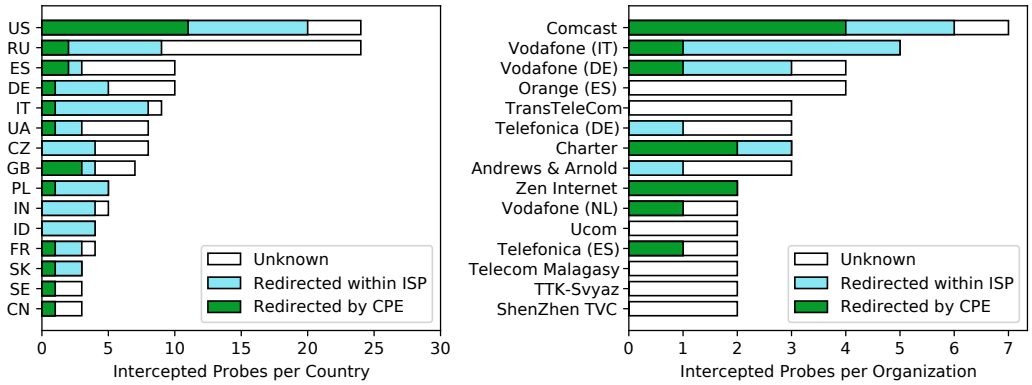

Figure 4.4. Interception location for the 15 countries and organizations with the most intercepted probes.

return a DNS error status for at least one resolver, such as `SERVFAIL` (server failure), `NOTIMP` (not implemented), or `REFUSED`. Because these status codes are not timeouts, they have likely been sent deliberately by the alternate resolver, and are not transient errors. We therefore conclude that some interceptors may block certain public resolvers.

Figure 4.3 shows that Comcast (AS7922) has the highest number of intercepted probes of any organization. Although RIPE Atlas has a high proportion of probes in the U.S., and in Comcast's AS in particular, this finding is consistent with prior work that showed significant DNS interception occurring in Comcast's networks [142]. Not all probes in Comcast's AS (or any AS) are intercepted: we found that this is because specific models of CPE perform the interception. For details in Comcast's case, see Section 4.7.

4.6.2 Is the interception performed by the CPE?

As we described in Section 4.3.2, once we identify intercepted probes, we use `version.bind` queries to determine whether the interceptor was the probe's CPE. Figure 4.4 shows the number of probes that were intercepted by their CPE per country and organization. To our surprise, a sizable fraction of the interception we observed was attributable to CPE: the CPE was the interceptor for 49 out of the 220 intercepted probes. Furthermore, we found such probes in countries around the world: these results are not due to an individual network's behavior.

Table 4.5. Strings sent in response to `version.bind`.

`version.bind` **Response**	**# Probes**
dnsmasq-*	23
dnsmasq-pi-hole-*	8
unbound*	6
-RedHat	2
PowerDNS Recursor*, Q9-*, 9.16.15, -Debian, Windows NS, Microsoft, new, unknown, none, huuh ? ... ,	1 each

* indicates version number

We grouped the `version.bind` responses from these 49 CPE-intercepted probes by the strings they returned, as shown in Table 4.5. The majority were various versions of Dnsmasq, software that is explicitly designed to run on CPE [77]. We consider Dnsmasq's presence to be confirmation that the interceptor answering the `version.bind` query is the CPE. We also saw Dnsmasq strings on eight probes that indicated the device was a PiHole, suggesting that the owner deliberately intercepted DNS (presumably to avoid advertisements).

4.6.3 Is the interception within the client's ISP?

If we are unable to confirm that the probe's CPE is the interceptor, we then check whether the interceptor is within the probe's ISP using *bogon queries* (Section 4.3.3). Figure 4.4 shows the number of probes that are intercepted by their CPE, intercepted within their ISP, and the probes whose interception location is (potentially) beyond the ISP. For the RIPE Atlas nodes at least, this technique finds that when DNS queries are intercepted, they are intercepted close to the client (at the CPE or ISP) in a majority of cases. Moreover, in many of the countries, interception more often than not happens within the ISP, matching prior work's findings [120, 142] that DNS interception is often a result of ISP policy.

4.7 Case Study: XB6 Router

We first started to investigate CPE-based interception when it began interfering with our previous DNS experiments. We first experienced interception when we discovered that one author could not contact public resolvers from her residence. Upon investigation, we were able to identify the router model that performed the interception: the Arris/Technicolor XB6 [200].

The XB6 is manufactured by both Arris and Technicolor, but its hardware was designed by Comcast [46]. It uses a firmware package called RDK-B (Reference Design Kit), which is in use by more than 80 million devices around the world [59]. Other ISPs also license RDK-B and rent XB6 routers to their customers, including Shaw Communications, Vodafone, Liberty Global, and many others [59].

Notably, RDK-B includes a DNS resolver called XDNS, which stands for Xfinity DNS [58]. XDNS can redirect DNS queries using DNAT, which Comcast uses to implement malware filtering services [60]. XDNS also implements a response to `version.bind`.

The XDNS filtering service is intended to be opt-in. However, it appears that a bug in some XB6 routers is causing them to direct all queries to the ISP's resolver, without giving users any indication that their choice has been curtailed. This problem is not limited to Comcast: we have observed very similar behavior in other networks where the XB6 is deployed, including Shaw Communications and Vodafone. However, the bug appears to be uncommon.

We have reached out to ISPs about these discoveries. Their responses have been supportive and we are working to identify the source of the bug.

4.8 Limitations and Future Work

This section describes our work's limitations and identifies directions for future work. Our method is designed to measure the systematic interception of *all* DNS queries sent to a target resolver. We looked for systematic interception since we had observed DNAT-based transport/network-layer interception in the wild (Section 4.7). However, if only some queries

are intercepted and others (such as our location queries) are not, our method will not determine interception.

Another limitation of our approach is that it relies upon the CPE answering `version.bind queries`. We do not expect this requirement to be a major limitation, however, since the BIND-like interface is now supported by many resolvers—even ones that do not use BIND.

Our approach also assumes that the DNS infrastructure of the client's ISP is located within the client's AS. If the ISP's resolver is located outside the client's AS, our approach will classify the interception's location as "unknown" instead of "within the ISP."

Additionally, our methodology may misclassify a non-intercepting CPE as an interceptor in a specific case: when the CPE has port 53 open, the CPE is a DNS forwarder, *and* the CPE does not respond to `version.bind` but instead forwards the query to a resolver.

We also note that RIPE Atlas is not a representative measurement platform and we therefore do not expect our results on the *prevalence* of DNS interception to generalize; we refer interested readers instead to recent work by Liu *et al.*, who investigated the prevalence of DNS interception [142]. Our goal in using RIPE Atlas is primarily to conduct a pilot study to show that our technique can detect interception and identify where it occurs.

While our approach should theoretically detect DNS interception in DNS over TLS (DoT) [110], we did not evaluate it on RIPE Atlas. DNS over HTTP (DoH) [107] and some configurations of DoT will prevent interception from occurring altogether, but the "opportunistic privacy profile" of DoT disables client certificate validation, so this configuration could allow interception. We leave evaluation of our method for detecting DoT interception for future work.

Techniques based on increasing the TTL of the IP header have the potential to identify which hop intercepted a query. The RIPE Atlas platform does not currently offer the ability to adjust the TTL of DNS requests, but we briefly explored using the VPNGate measurement platform [212] for this purpose. Unfortunately, we found that their VPN rewrites IP TTLs, rendering this experiment impossible. However, changing non-ICMP packet TTLs requires root or SUID root on most systems, whereas our approach only requires the ability to send DNS

queries.

4.9 Related Work

Due to the vital role of recursive DNS resolvers in Internet interactions, many of their properties have been studied, including their proximity to clients [30, 150, 166, 62, 34], their caching behavior [122, 121, 157, 129], and home gateway behavior [104]. Prior work has also studied vulnerabilities affecting DNS resolvers [197, 190, 106] and DNS cache snooping side channels that can reveal the popularity of web domains, as discussed in Chapter 3 and several other studies [226, 179, 180, 151]. We focus our discussion on work that has studied DNS interception and DNS redirection.

Most prior work has studied *DNS redirection*, where (some) DNS responses received by user devices are altered (Section 4.2). DNS redirection is often employed to implement country-level or ISP-level policies. Researchers have reported that DNS requests sent from within China to third-party resolvers outside the country face DNS injection [41] (likely to implement censorship measures), and that even DNS requests that originate outside China but transit the country can be redirected due to collateral damage [40]. In a similar vein, studies have leveraged open recursive resolvers to investigate DNS manipulation whose likely purpose is to restrict user access to content [171, 126]. Chung *et al.* reported on NXDOMAIN wildcarding—the practice of rewriting NXDOMAIN errors with A records that point to a web server—and show that this form of DNS redirection may be occurring at the ISP's DNS server, public DNS servers, and ISP middleboxes [65]. Using data from Netalyzr, Kreibich *et al.* showed that NXDOMAIN wildcarding practices were prevalent among several ISPs [135, 217, 218]. Complementing these findings, our study shows that some instances of DNS *interception* occur due to potentially misconfigured CPE infrastructure; replacing these CPE devices sometimes suffices to prevent DNS interception. Our study also differs from previous work on NXDOMAIN wildcarding in that we study transparent interception rather than interception that is detectable by the client.

Also using RIPE Atlas probes, Jones *et al.* and Wei *et al.* measure how frequently DNS debugging queries (specifically `hostname.bind`) towards root servers are redirected [120, 222]. Their discovery of DNS redirection in existing RIPE Atlas datasets encouraged us to use the platform for our pilot study on DNS interception, although our work found `version.bind` to be better suited for our purposes.

Vallina-Rodriguez *et al.* measure the prevalence of DNS proxies in cellular networks by sending queries to their own authoritative nameserver [39]. Our work differs since we focus on where in the network interception is happening instead of its prevalence.

Liu *et al.* measured the prevalence of DNS interception in a recent study [142]. They performed DNS requests over both a commercial proxy network and from several Chinese mobile networks to estimate how common DNS interception is, but did not investigate where in the network interception takes place.

To the best of our knowledge, we are the first to investigate DNS interception over IPv6. Our results on RIPE Atlas suggest that DNS interception occurs far less frequently in IPv6 than in IPv4.

4.10 Summary

This chapter provides a methodology for identifying the location of a DNS interceptor, whether the interceptor is the host's CPE device, a device in the host's AS, or elsewhere. Being able to empirically determine such information — that would otherwise be invisible to the user — is particularly relevant now when concerns about privacy and integrity have led to multiple proposed standards for embedding DNS traffic within encrypted tunnels: DNS over HTTPS and DNS over TLS. While the complex and many-sided nature of the debate around the deployment of such protocols is beyond the scope of this short paper, it is motivated by precisely the kinds of DNS interception that we observe and that can be more closely monitored by using our work.

Chapter 4, in part, is a reprint of the material as it appears in *Proceedings of the Internet*

Measurement Conference 2021. Audrey Randall, Enze "Alex" Liu, Ramakrishna Padmanabhan, Gautam Akiwate, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Challenges of Blockchain-Based Naming Systems for Malware Defenders

5.1 Overview

This chapter describes how we use a naming system to not only measure an adversary's behavior, but also design effective interventions against that adversary. We also introduce the second type of naming system that we examine in this work: blockchain-based naming systems, or BNSes. The adversaries we study in this chapter are are malware operators. Malware that is distributed across multiple hosts needs a way to distribute commands, upload stolen data, and coordinate between infected machines. Most malware, such as botnets or ransomware, uses a central command and control (C2) server for this task. However, as a single point of failure, a central C2 server presents an obvious weak link for defenders to target [125]. Malware authors must therefore be able to easily relocate and replace a C2 server after a defender takedown. Furthermore, all previously infected hosts must be able to find the new server at its new address, without outside coordination — if they cannot, they become useless. Malware authors avoid this "sunk cost" problem by providing a layer of indirection — a *naming layer* — instead of hard-coding a fixed address directly into deployed malware. This naming layer must be resilient to takedown efforts.

Until recently, the naming layer used most frequently by malware was ordinary DNS, which is rarely blocked at the protocol level, universally supported, and easy to configure.

Malware authors use various strategies, such as DGAs (domain generation algorithms), to cycle through domains and complicate defense efforts. However, DNS domains are subject to centralized authorities such as registrars, who may be compelled to seize or deny access to abused domains. Malware authors have recently come up with an innovative solution to this risk: they have started to use *blockchain-based naming systems*.

Blockchain naming systems present several potential challenges for defenders. First, because they have no central authority to carry out legal takedown requests, they are immune to one of the most effective tools in malware defenders' arsenals. Second, some blockchain naming systems have high transaction costs to register and manage domains, which renders some existing defense strategies ineffective. For example, registering all the domains that a DGA can generate is impractical in expensive blockchain naming systems. However, blockchain naming systems present challenges to malware authors as well, such as the difficulty of stealthily accessing the blockchain.

We study five blockchain naming systems and the challenges and advantages that each present to malware authors and defenders. We argue that defender interventions are still possible for each of these systems, because name resolution requests must pass through centralized or partially centralized infrastructure to access any blockchain naming system. These centralized "chokepoints" still present viable locations for defenders to stage interventions. We also perform a measurement study of how malware is currently using these naming systems, and conclude that while some systems have seen significant abuse, others are currently much less likely to attract malware due to their high cost. We conclude that while blockchain naming systems present a significant threat, defenders still have viable options for enacting C2 takedowns.

5.2 Background

From a malware author's perspective, an ideal naming system for C2 addresses should have two properties: it should be difficult to censor specific individual names, and it should also

be costly to take down or block the system as a whole. Typically, this first property, per-name takedown-resistance, is directly tied to the existence of a central authority with the ability to take down an individual record. System-wide takedown-resistance, on the other hand, is commonly a byproduct of a system's overall utility to legitimate users; potential collateral damage to benign users can create strong pressures against such takedowns.

To some extent, a trade-off exists between these features. For example, protocols such as Tor provide high resistance to the censorship of specific names, but because Tor traffic is both easily identified and represents a small volume of users, some defenders will simply block Tor altogether (i.e., under the assumption that any damage to benign users will be minor). On the other side of the spectrum, malware has repurposed ubiquitous systems such as social media to store C2 addresses, because such traffic does not stand out and blanket bans on social media URLs are practically untenable. However, social media companies such as Facebook and Twitter have the capability, motivation, and (at times) legal obligation to remove abusive posts used to coordinate C2 activity. Thus, malware authors are incentivized to find naming systems that are neither vulnerable to censorship of individual records nor likely to be blocked or taken down entirely.

5.2.1 Tradeoffs of DNS-based C2 names

In part due to its ubiquity, DNS has been one of the most widely used naming systems for malware C2 servers. Because of its central role in the Internet's function, DNS naturally satisfies the system-wide takedown-resistance requirement; completely blocking DNS would be unthinkable for any modern enterprise or ISP. However, DNS is subject to a hierarchy of defined authorities, each of which may disable individual domain names, either voluntarily or in response to legal compulsion. For example, the registrar that sold a domain can delete it, or sinkhole it (i.e., divert its traffic) and prevent it from being subsequently updated or transferred. Similarly, the registry responsible for a domain name's top-level domain (TLD) can also take such actions.

Key to all these actions is that there is a singular legal entity with the capability to intervene. In some cases, they may do so voluntarily (e.g., such as when a registrar is notified of a violation of their terms of service.[1]) but they may also be compelled to take action by a court order. For example, US law enforcement may obtain a warrant to seize control of a domain name (subject to a showing of probable cause that it is involved in a criminal act), so long as the controlling registrar or registry is within US jurisdiction.[2] Typically such names are not then deleted (which would allow the malware author to re-register them), but instead the registrar (or registry) is compelled to redirect traffic to a benign site (aka a sinkhole) and block any attempts to change or transfer the domain by the registrant. Civil litigants can obtain similar effects via Temporary Restraining Orders (TRO) typically based on a showing that their intellectual property rights are being violated [125]. Microsoft, in particular, has made innovative use of trademark protection laws, such as the Lanham Act, to drive expansive private-sector botnet takedown efforts [138].

Today such legal takedowns are a critical tool for defenders to disable botnets. However, the effectiveness of this tool has led malware authors to respond by developing Domain Generation Algorithms (DGAs) which minimize the impact of any given takedown. When an infected host uses a DGA, it can randomly generate a large number of domains that its C2 server might be found at. DGAs usually use the current date as an input, which allows them to be kept in sync and changed on whatever time scale is convenient for the malware operators. Only a few of these domains will be registered at a time, to prevent defenders from preemptively taking them down. If a domain that is currently used to contact the C2 server is seized, the malware operators simply register a new one. Upon failing to reach the old C2 domain, every infected host will begin trying to resolve the rest of the names generated by the DGA until they discover the new, working C2 domain. Thus, to truly take down such a C2, defenders must register all of

[1] Registrars and hosting providers all typically specify a "Acceptable Use Policy" (AUP) in their contracts with third-parties. The details of these AUPs vary between providers, but it is common that they prohibit activity that is criminal, that violates intellectual property interests, or is highly disruptive.

[2] This process is described in considerable detail by Knight [130].

the domains (or otherwise prevent them from being registered) that the DGA can generate [42]. This intervention is costly in effort, but has been used successfully in the past [173].

To summarize, DNS-based C2 represent the status quo for modern botnets, but are vulnerable to concerted legal takedown efforts. What makes these takedowns possible is the existence of singular entities with the capability to unilaterally assert control over individual names *and* that those same entities are subject to legal jurisdictions available to defenders. Malware authors are thus incentivized to find naming systems that are not vulnerable to such efforts, either because the authority that can control the names is not in a takedown-amenable jurisdiction, or because no such authority exists.

5.2.2 Blockchain-based domain names

Blockchain-based naming systems present a potential threat because they claim to be immune to takedowns. This supposed immunity stems from several factors. First, no central authority controls blockchain domains in the same way that registrars control traditional DNS names. Unlike a DNS record, it is not generally possible to modify or delete a record on a blockchain without controlling the record's private key. Once a domain has been registered, its ownership is passed to the purchaser, after which point even the company that sold it cannot modify it.[3] Second, the machines running a blockchain are often distributed across so many countries and jurisdictions that seizing or taking down the entire system is prohibitively impractical. Third, once created, name records are stored immutably on the blockchain for as long as that blockchain exists, even if the owner later modifies or deletes them. With sufficient resources, and assuming all data is stored on-chain, it is possible to reconstruct the value of a blockchain-based naming record at any point in time, by parsing the transactions that modified the record's state up to that point.

These censorship-resistant properties are generally true under the assumption that an

[3]This is true except in the case where the seller provides a domain parking service - we discuss this case in Section 5.4.6.

adversary has not found a way to compromise the entire blockchain, e.g. by gaining control of more than half of the blockchain's computational power. Such attacks are generally extremely difficult to execute. We focus instead on the common case where the blockchain underlying a naming system is not compromised, in which case the naming system as a whole is highly resistant to takedown efforts. Furthermore, some blockchains have become popular enough that even blocking access to them at a network level would cause collateral damage to licit users. Blockchains such as Bitcoin and Ethereum have recently skyrocketed in popularity as investors became interested in cryptocurrency as an asset class. As far as we are aware, cryptocurrencies and the blockchains they rely on are the first examples of strongly censorship-resistant systems that have gained a substantial community of legitimate users around the world.

Blockchain-based naming systems therefore provide both desirable properties of naming systems for C2 servers: it is difficult to take down the whole system as well as to take over individual records. Unfortunately, some malware is already aware of these advantages. BazarLoader uses Emercoin to record the domains of its C2 servers [52]. Namecoin is used by the Necurs botnet [18], the Chthonic banking trojan [21], Smoke Loader/Dofoil, Backdoor.Teamviewer, Shifu, and TinyNuke [1, 148]. Cerber ransomware has even used blockchain wallet addresses as names for its C2 servers [175].

5.3 Overview of Blockchain Naming Systems

In this section we present an overview of five blockchain-based naming systems, to provide background on how such systems work in detail. We select these systems based on their apparent popularity, as well as prior reports and literature that indicate some of them have already been abused by malware. These naming systems fall into two categories: systems built on naming-specific blockchains like Namecoin and Emercoin, whose purpose is primarily to store names and records, and systems built on general-purpose blockchains such as Ethereum, that are designed for purposes beyond naming systems. These systems also fall into two "generations:" Namecoin

Table 5.1. Non-exhaustive selection of proxies, browsers, and extensions that can be used to access blockchain-based naming systems.

Name system	TLDs	Proxies
Namecoin	.bit	BDNS (defunct), PeerName
Emercoin	.lib, .bazar, .coin, .emc	friGate, PeerName, OpenNIC
Handshake	*any string*	hns.to, NextDNS, HDNS.io, BobWallet extension, LinkFrame extension
ENS	.eth	eth.link, eth.limo
Unstoppable	.crypto, .blockchain, .bitcoin, .coin, .nft, .wallet, .888, .dao, .x, .zil	Brave browser, Opera browser, Unstoppable browser, Unstoppable extension, Infura

and Emercoin have existed since 2011 and 2013 respectively [10, 12], while the Ethereum Name Service (ENS), Handshake, and Unstoppable Domains are more recent inventions (2017, 2018, and 2019, respectively [6, 8, 87]).

All of the blockchain-based naming systems we study differentiate their names from DNS domains by creating alternate top level domains, which we refer to as *alt-TLDs* for brevity. A summary of the alt-TLDs used by each naming system is presented in Table 5.1. Handshake names are slightly different, since the goal of the Handshake project is to replace the DNS root zone and make any alt-TLD available for purchase — see Section 5.3.1 for more details.

5.3.1 Naming-specific blockchains

We study three naming systems that are built on naming-specific and eponymous blockchains: Namecoin, Emercoin, and Handshake. All of these blockchains are primarily designed to support their naming systems, rather than to create new cryptocurrencies or support arbitrary blockchain-native programs ("smart contracts"). Because these blockchains have such specific purposes, they differ from blockchains like Ethereum in two ways: they have fewer participants and users, and their transaction fees are much less expensive. Both properties have implications for defenders — see Section 5.4 for more details.

Namecoin and Emercoin

Namecoin and Emercoin, which are both modified copies of Bitcoin, are the oldest blockchain-based naming systems. Both were intended as additions to traditional DNS: users registered domains that resolved to IP addresses, using records very similar to DNS records. Unfortunately, Namecoin and Emercoin have been subject to a large amount of abuse. Four years after Namecoin's launch, Kalodner et al. found that only 28 of the 120,000 domains registered in Namecoin had meaningful web content, and most domain registrations appeared to be squatting [123]. In 2021, Casino et al. collected all of the IP addresses stored by Emercoin and Namecoin records, and submitted them to threat intelligence services including VirusTotal,

Table 5.2. Record types in the Handshake namespace.

Record	Names with Record
Default NS and GLUE4 records	102,386
No A records	102,285
A 44.235.163.135	94
A 52.43.158.89	4
A 144.91.114.245	2
A 1.1.1.1	1
Invalid name	98,068
No record (null)	845
TXT record	138
"hello fx-wallet"	110
Other	28
Non-default NS record	32
Non-default GLUE4 record	11
Distributed storage address	7
Total unique names	201,458
Total records	201,487

Hybrid Analysis, Abuse.ch, and Pydnsbl (an aggregator of blocklists). They found that over 50% of the IPs in Namecoin and Emercoin records had been flagged as malicious by at least one threat intelligence service [56]. Furthermore, Casino et al. used a "poisoning" approach to find IP addresses associated with malicious IPs, either because the two addresses were stored in the same wallet, a name resolved to both addresses at different times, or the same email was recorded in their records. This "poisoning" approach revealed that the vast majority of IP addresses in Namecoin and Emercoin records are connected in some way to malicious IPs [56]. Our own findings support the conclusion that these naming systems are still rife with abuse (Section 5.5).

Handshake

Handshake is a blockchain-based naming system that aims to replace the root DNS zone. It offers its users the ability to purchase nearly any string to use as an alt-TLD. Rather than selling second-level domains itself, Handshake allows its users to act as registrars who can sell their own domains. Handshake records are designed to store the NS records of traditional

authoritative nameservers, rather than to replace DNS A, AAAA, or similar records. Handshake also allows users to store TXT records, which can contain the addresses for decentralized web hosting systems like Skynet [5] or IPFS [9]. Malware operators could potentially use Handshake as a naming system to find C2 content stored in these distributed storage systems. Additionally, Handshake advertises themselves as "the only naming blockchain with a lightweight recursive DNS resolver, which you can easily embed into browsers, apps, and devices" [2]. This lightweight resolver may be attractive to malware operators because it is small enough to be part of a malware payload.

To get a sense for how people use Handshake, we collected a sample of approximately 201,000 recently registered Handshake names by scraping a Handshake block explorer.[4] We attempted to scrape these names directly from the Handshake blockchain, but were unsuccessful because the RPC provided by the Handshake client to collect registered names from a Handshake node is no longer functional [11]. Table 5.2 summarizes our findings. At the moment, Handshake names appear to be overwhelmingly utilized as speculative assets. Only 0.14% of names in our sample had NS records that differed from the default. Of the names that kept the default nameserver and glue records, only 101 (0.05%) eventually resolved to A records, 94 of which were for the same IP address (a nameserver run by Namebase). Nearly half of registered Handshake domains in our sample cannot be resolved by the HNS client, since they contain illegal characters like emojis or are solely composed of numbers: these names are nevertheless allowed to be created on the Handshake blockchain. We concluded that the Handshake system has not yet seen significant adoption by either licit users or malicious actors.

5.3.2 Naming systems on general purpose blockchains

Two naming systems based on the Ethereum blockchain have arrived since 2017: the Ethereum Name Service (ENS) and Unstoppable Domains. These naming systems are possible because of Ethereum's innovation in the blockchain space: *smart contracts.* Smart contracts

[4] https://e.hnsfans.com/names

are code that is embedded into the Ethereum blockchain. Any machine that runs an Ethereum "full node" can execute any smart contract. Each contract is identified by a 20-byte address, and makes its functions available through its Application Binary Interface (ABI). Thus, asking a smart contract to execute one of its functions is similar to making an RPC call, except that instead of one machine executing the code, every machine that receives the transaction must do so.

Smart contracts can be used to implement key-value stores, which means they are well suited to act as naming systems. For example, in a simplified system, a user might wish to set the name "foo.crypto" to resolve to the IP address 1.2.3.4. The user would create an Ethereum transaction that asks the key-value store's smart contract to call its "set record" function, with "foo.crypto" and "1.2.3.4" as function inputs. This transaction is then broadcast to the Ethereum network, and every Ethereum node that receives it updates its own copy of the key-value store to include the new record. Reading from the key-value store works similarly to writing to it: any Ethereum node can return a correct response. Notably, any transaction that causes a write costs a "gas fee" of Ethereum cryptocurrency. Gas fees are dependent on network congestion as well as other factors: they incentivize Ethereum node operators to execute smart contract code, which uses computing resources. In contrast, reading a smart contract's data does not cost a gas fee and does not create a transaction.

Interestingly, ENS and Unstoppable Domains are structured like DNS, but they are not necessarily being used as DNS replacements. The language and structure of both systems' smart contracts implies that they were modeled after DNS: for example, both systems use certain smart contracts as registries, and ENS even uses others as resolvers and registrars. However, users are primarily using these systems to map human-readable names to *cryptocurrency wallet addresses* instead of IP addresses. While users can still store IP addresses, traditional domains, TXT records, or distributed storage system (DS) addresses, very few choose to do so. This may imply that C2 records containing IP addresses or DNS domains will stand out and be easier for defenders to detect.

Table 5.3. The ENS resolvers from which we collected a sample of names and records.

Resolver Name	Txns Setting Resolver	Address
Public Resolver 2	33,304	`0x4976fb...`
Public Resolver 1	2,736	`0xDaaF96...`
OpenSea ENS resolver	482	`0x9C4e9C...`
ENS Old Public Resolver 2	440	`0x226159...`
Umbra: Stealth Resolver	409	`0xB37671...`
unnamed PublicResolver #1	126	`0xD3ddcC...`
unnamed PublicResolver #2	103	`0x5FfC01...`
ENS Old Public Resolver 1	29	`0x1da022...`

ENS

ENS names are registered (and resolved) in two steps involving two different smart contracts. First, a name must be registered using the "ENS Registrar Controller" smart contract, which accepts the human-readable name and the address of a contract to use as a "resolver." Second, the resolver contract must be updated with the name's records. To complicate matters, names are not handled in their human-readable forms after they are registered: instead, they are referred to by their keccak256 hash. Furthermore, the ENS Registrar Controller contract allows users to specify a hash instead of a human-readable name, without ever performing a transaction that reveals the name itself. Therefore, to enumerate most of the names in ENS, we had to parse all of the transactions in the ENS Registrar Controller contract that recorded new hashes of names. We then queried the associated resolvers to discover the human-readable names. At the time of writing, at least 504 smart contracts had been set as resolvers for at least one name hash. We chose to take a sample of names from the eight resolver contracts that were set by the most names as their default resolver. The distribution of resolvers is long-tailed: the majority of resolvers resolve only a few names, while the eight most popular resolvers resolve the majority of names. We excluded addresses that were set as resolvers by many names but did not implement the ENS resolver specification, under the assumption that these were mistakes. Such misconfigured resolver addresses include the null address, 0x0, as well as unrelated smart

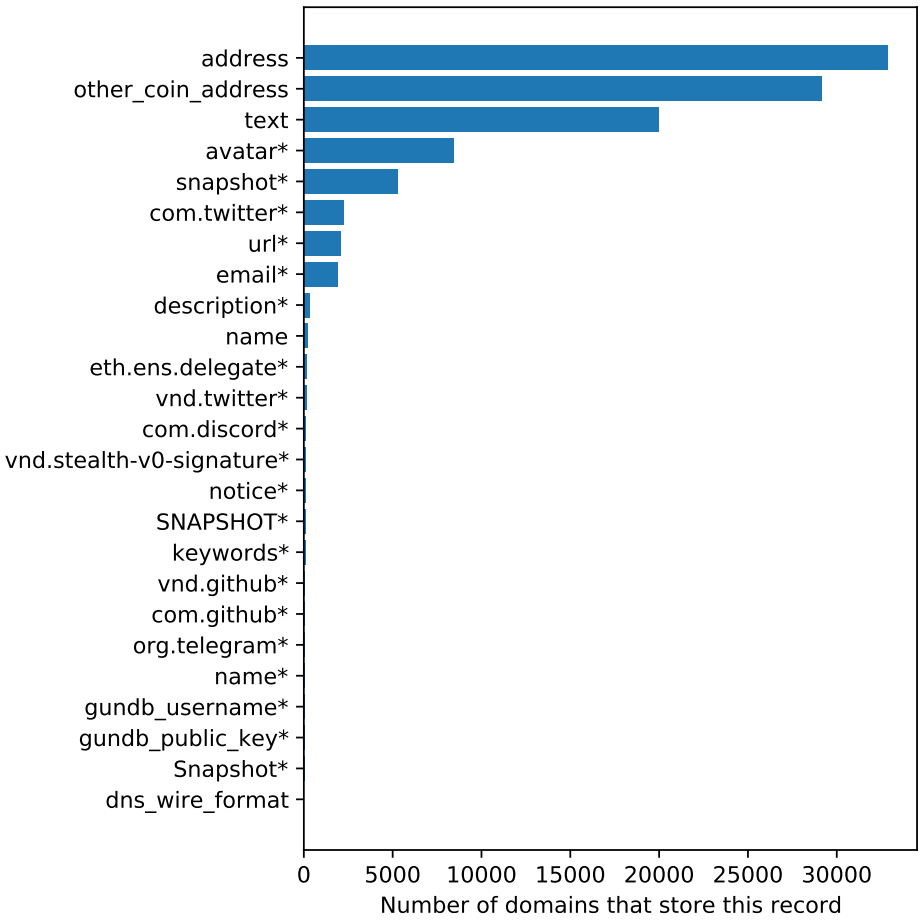

Figure 5.1. Records stored by ENS names. *key within "text" record

contracts used by the ENS ecosystem. The resolvers we chose are detailed in Table 5.3. This approach yielded a sample of 667,369 ENS names that were registered through the ENS Registrar Controller contract. Prior work has found that even after collecting all transactions from the ENS Registrar Controller and its historical predecessors, some hashes appear in the system but have never been seen to resolve to names [228]. It is unclear how these hashes came to exist, so we note that our sample does not contain all of the names in ENS, just the majority.

Figure 5.1 shows the distribution of the types of records stored in ENS for our sample of names. The majority resolve to wallet addresses or text records, not IP addresses, traditional domains, or DS addresses. We broke down the text records, which are key/value pairs, by the most common key names: these keys are marked with an asterisk. Only the most common 25

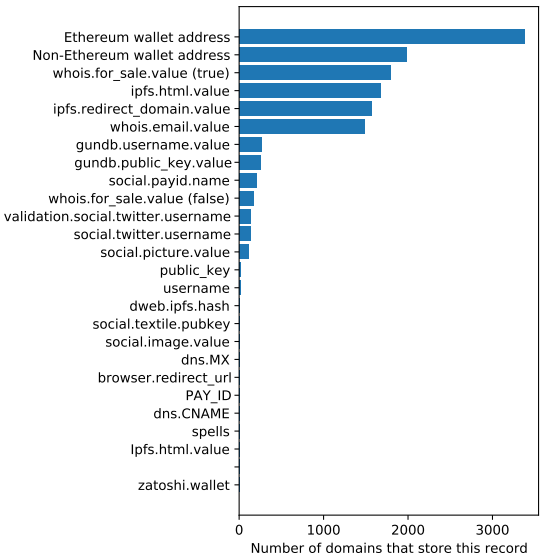

Figure 5.2. Records stored by Unstoppable Domains names.

keys are shown. We note that only 17 names had `dns_wire_format` records, which are intended to store traditional DNS records, and all 17 are malformed as far as we can tell.

Unstoppable Domains

Like ENS, Unstoppable Domains uses Ethereum smart contracts as registrars. Unstoppable Domains names are divided into two systems. CNS (the Crypto Name System) contains all names with `.crypto` alt-TLDs, and has separate registry and resolver contracts. Later, Unstoppable added UNS (the Unstoppable Name System), which simplified name resolution by combining the resolver and registry contracts, and added several new alt-TLDs. Unstoppable Domains names never have to be renewed; they are purchased once and then owned indefinitely.

Like ENS names, Unstoppable Domains names are referenced by their hashes. We extracted all hashes from the UNS and CNS registry contracts by searching all of their transactions, and then found each hash's name and records by querying Unstoppable Domains' metadata endpoint.[5] This approach yielded a sample of 16,026 names. As with ENS, some names appear to exist in Unstoppable Domains that cannot be found by collecting transactions from

[5] `https://metadata.unstoppabledomains.com/metadata/`

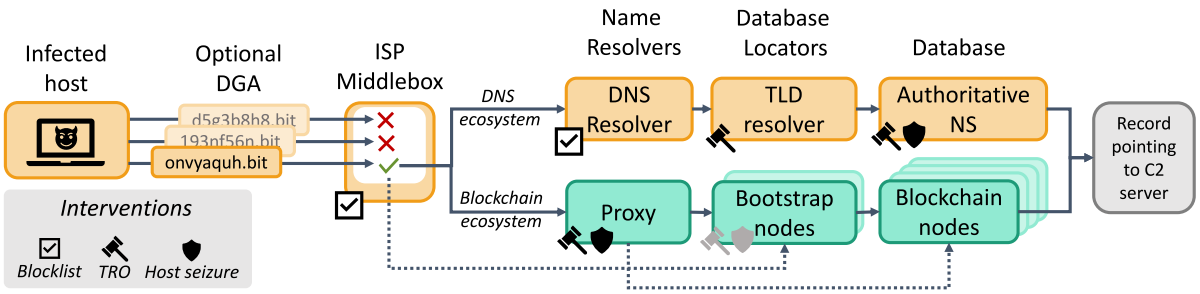

Figure 5.3. Potential locations of interventions for blocking access to DNS-based and blockchain-based C2 server names.

these registry contracts. For example, it appears to be possible to store Unstoppable Domains names on the Polygon blockchain instead of Ethereum. We therefore note that our sample does not contain all of the names present in Unstoppable Domains.

Figure 5.2 shows the distribution of record types found in the Unstoppable Domains names. As in ENS, the majority of names have wallet records rather than records that point to websites in any way. The second most common type of record is "whois.for_sale.value," showing that many names are seen as speculative assets. Unstoppable Domains also provides an easy way for users to link to IPFS records.

We performed a web crawl of all of the Unstoppable names that had records pointing to websites, whether IPFS records, traditional IP addresses, or traditional domains. We took screenshots of the 367 websites we arrived at, inspected them manually, and did not find any evidence of malware use. Most websites were personal sites, Web3-based business sites, or related to the sale or collection of NFTs.

5.4 Intervention Locations

Accessing any naming system, whether blockchain-based or DNS-based, requires a number of steps, each of which presents an opportunity for defenders to stage an intervention. Figure 5.3 compares the steps an infected host takes to resolve a DNS C2 domain (shown in orange) to the steps required to resolve a blockchain name (shown in green). While these

steps involve different participants, parallels exist between the blockchain and DNS ecosystems. Figure 5.3 also details the interventions that can be staged by defenders at each step of the process. We now describe each step and its potential interventions in detail. For certain interventions at certain steps, we also present a case study of an attempt to stage the intervention in the wild, its results, and the lessons it provides for defenders.

5.4.1 Reaching the resolver

Regardless of whether a request is destined for DNS or a blockchain naming system, it must first reach the machine that acts as a resolver: either the DNS resolver or the proxy in Figure 5.3. Defenders may be able to intervene before this point by placing middleboxes with filter lists in the network. Some networks already have such defenses: for example, some ISP networks redirect all DNS requests to the ISP's own resolver, which can implement a filter list. This defense is probably not currently intended to block blockchain names, but it has that effect nevertheless in some cases. For example, some malware uses ordinary DNS rather than DNS-over-HTTPS to request blockchain domains, under the assumption that the proxy the query is intended for will redirect it to the blockchain naming system in the correct format. When ISPs perform DNS redirection to their own resolvers, these queries get redirected to the DNS root, which cannot resolve the alt-TLDs used by blockchain naming systems and return "NXDOMAIN." We present our study of this phenomenon in Section 5.5. We observe that filter lists are only a partial defense against malware, because malware may utilize DGAs to evade them: as soon as a C2 name is added to the blocklist, the malware operators may register and begin using a new one.

5.4.2 Interventions at the name resolver

When resolving DNS domains, the entity that first attempts to respond to the request is a DNS resolver. When using blockchain naming systems, this entity is a proxy instead, shown in Figure 5.3 under "Name Resolvers." The proxy may expect queries in the form of DNS-over-

HTTPS, unencrypted DNS, or in an arbitrary format. Instead of querying the DNS root zone, the TLD resolver, and eventually the authoritative nameserver to resolve a name, the proxy must connect to the blockchain and retrieve the record from one of the participants. Defenders may intervene at a traditional DNS resolver by requesting that the resolver implement a filter list, or the resolver operators may elect to implement one voluntarily. However, proxies that resolve blockchain names may be resistant to such voluntary efforts, because the blockchain ecosystem is often organized around principles of independence and self-governance, and resistance exists to the idea of censoring any content.

Proxies are currently the most common method for resolving blockchain names. Table 5.1 shows a selection of the proxies and tools that resolve names from each of the systems we study. The list of proxies is not exhaustive, but represents a subset of the best-known proxies in use at the time of writing. While most large browsers, such as Safari, Chrome, and Firefox, do not support any blockchain naming systems natively, some naming systems provide browser extensions that redirect blockchain name queries to proxies using DoH. A few browsers do resolve blockchain names without requiring extensions, such as Brave, which partners with a proxy called Infura [16]. Some naming systems have partnerships with existing DNS resolvers. For example, NextDNS's DNS resolvers can act as proxies to resolve Handshake names. Finally, some naming systems, such as Handshake, also provide stub resolver implementations that run locally on a user's computer. These stub resolvers also work by routing blockchain name queries to proxies.

Almost all of these proxies are centralized, in that they are controlled by a single authority. This is good news for defenders: similarly to traditional registrars, they are vulnerable to legal takedowns. They can be served with TROs or warrants and compelled to stop giving access to abused domains, as long as they operate within a jurisdiction amenable to such efforts. A centralized proxy could also be neutralized by serving a takedown order to its hosting provider, although this approach would produce varying amounts of the collateral damage depending on how many licit users utilize the proxy. While these interventions are not foolproof, they

are subject to the same advantages and disadvantages as interventions on traditional registrars. Thus, centralized proxies return the distributed naming ecosystem to a state similar to the DNS ecosystem, from a defender's point of view.

Case Study #1 — OpenNIC ceasing support of `.bit`

OpenNIC is one of the few decentralized proxy services for blockchain names. It resolves names from several alternative naming systems, including Namecoin and Emercoin [13]. OpenNIC's resolvers are run by a small community of volunteers [14]. In June 2019, this community voted of their own volition to remove support for Namecoin's `.bit` alt-TLD, because providers were beginning to block OpenNIC resolvers that were used by malware to resolve Namecoin names [17].

OpenNIC's decision to cease supporting Namecoin was not the result of a direct intervention by defenders, but it still yields an important lesson. Even a decentralized proxy service may be composed of few enough individuals that it is possible to cajole or compel them to stop resolving names used by malware. Furthermore, OpenNIC's community held this vote in response to pressure from Spamhaus, Malwarebytes, and other providers, who began blocklisting OpenNIC resolver domains: even if a proxy's operators cannot be contacted directly, it is still possible to pressure them to cease resolving names used by malware.

Case Study #2 — BDNS takedown

In April 2021, various defenders attempted to take down a proxy known as "Blockchain-DNS.info" or BDNS. BDNS reported on their website soon after the takedown attempt that seven of their domain names had been "un-delegated" and one of their API servers was shut down without warning [23]. For example, one endpoint, `bdns.io`, was apparently sinkholed by ShadowServer [19]. `bdns.io`'s NS records now point to variants of the name `sinkhole.shadowserver.org`.[6] We confirmed that these NS records were changed to

[6] sinkhole-0[0-4].shadowserver.org and sinkhole-[a-b].shadowserver.org.

ShadowServer's domains on March 26, 2021, using the pDNS database [78]. BDNS received a message from Spamhaus shortly after noticing the takedown, stating that several of BDNS's endpoint domains had been added to Spamhaus's blocklist. BDNS claimed that their browser extensions continued to resolve blockchain names using other endpoints, and directed users to a list of endpoints that were still working [24]. BDNS also stated that they had moved some infrastructure to a friendlier hosting provider, PRQ, which states on its website that "If [content] is legal in Sweden, we will host it, and will keep it up regardless of any pressure to take it down" [26]. However, as of August 2022, all of the endpoints listed in BDNS's Github repository [24] are either failing to resolve or resolving but failing to load content, and the proxy appears to be nonfunctional.

This takedown effort provides several lessons for defenders. First, defenders must take care when choosing a takedown strategy for a proxy. In this case, defenders tried two tactics: adding the proxy's endpoints to a widely used blocklist and taking down some domains and a hosting server entirely. The former tactic appeared to work well in locations where ISPs use Spamhaus's blocklist: BDNS stated that their proxy "may be still unreachable in those parts of the world." However, the domain takedown appeared to be only partially effective, since BDNS could still resolve blockchain names for a time using unaffected endpoints. We conclude that care must be taken to enumerate all of a proxy's endpoints and shut them down simultaneously.

5.4.3 Skipping the proxy: the rise of light clients

While proxies greatly simplify the process of connecting to a blockchain, they are not strictly necessary, which is bad news for defenders. We initially assumed that no infected host would be able to skip the proxy and participate directly in the blockchain, because acting as a blockchain node requires too many resources. However, this assumption turned out to be incorrect, because of the rise of *light clients*. When blockchains were first envisioned, most assumed that every participant in the network would be a "full" implementation of a node: it would contain enough state to reconstruct the entire history of the chain, all the way back to the

first transaction. Additionally, each node would contribute to the blockchain by verifying every transaction it heard about. As blockchains grow over time, they become too resource-intensive to run on anything other than a dedicated, powerful machine. Two resources serve as the constraints: first, CPU power, which is obviously necessary to perform mining but now is even a bottleneck for transaction verification, because so many transactions happen per second. Second, disk space and speed: for example, a full Ethereum node cannot be run on a machine with a hard disk drive anymore, because nothing slower than a solid state drive can keep up with the reads and writes required [7]. These resource constraints make it very unlikely that malware could run "full" blockchain nodes on infected hosts. However, these constraints have also given rise to the concept of a "light client," a blockchain node with limited functionality that can fetch transactions from the chain but does not contribute by verifying transactions, mining, or broadcasting. Light clients are designed to run on laptops and mobile devices. As such, they use few enough resources to reasonably be included in malware.

5.4.4 Interventions at the database locator

Light clients enable malware to act as a first-class member of a blockchain, and discover other members of the chain using the chain's peer-to-peer discovery protocol without using a centralized proxy. In this case, defenders are left with a harder location to stage an intervention: the blockchain's bootstrap nodes, which is the blockchain equivalent of a service that locates the database of naming records. We show this path in Figure 5.3 with the dotted line between the middlebox and the bootstrap nodes.

In traditional DNS, the resolver must locate the database that contains a record by first querying the hierarchy of DNS servers: first the root and then the TLD resolver. The TLD resolver's role is to tell the DNS resolver which machine stores the database that ultimately contains a name's records. In a blockchain system, this role is filled by the bootstrap nodes. The purpose of the bootstrap nodes is to provide a gateway to the blockchain for new participants: new blockchain nodes find their initial list of potential peers by connecting to the bootstrap nodes.

Blockchains use various methods to publish bootstrap node addresses for their users. For example, Ethereum uses a list of bootstrap nodes that are hard-coded into client implementations [27]. Bitcoin stores lists of bootstrap nodes in DNS TXT records maintained by volunteers, as well as hard-coded lists [15].

When defenders perform interventions by putting legal pressure on registrars, the intervention takes effect at the TLD resolver, which implements the changes to the zone file that affect the malware's domains. These changes can include "sinkholing" the domain by causing it resolve to an IP controlled by defenders or "freezing" it so that its records cannot be modified. This intervention does not translate well to blockchain naming systems for several reasons.

First, while bootstrap nodes are responsible for finding the entire naming database, they do not allow defenders to specify which blockchain systems a client may access and which it may not. This means that seizing a specific naming record, or even the entire naming system, is not possible at the bootstrap nodes. Consequently, disabling or seizing bootstrap nodes prevents all new clients from accessing any functionality provided by the blockchain, including the blockchain's cryptocurrencies and any services it offers unrelated to naming. This approach therefore carries the potential for a lot of collateral damage. Second, bootstrap nodes may be widely distributed across the globe, leading to jurisdictional challenges in bringing legal pressure to bear on their operators. Bootstrap nodes may also be difficult to find, since they may not be run by hosting providers but rather by anonymous individual volunteers. Third, bootstrap nodes may be numerous enough that finding and seizing them all may be prohibitively difficult. Finally, while the default bootstrap node lists are published for each blockchain, users may choose to substitute their own. A malware author could design a payload that contains an extensive list of machines that participate in a blockchain naming system, which would complicate a defender's efforts to take down all the potential participants. Because interventions at the bootstrap nodes are more challenging than interventions at the proxy, we show the intervention icons in gray in Figure 5.3.

However, defenders could fall back to using blocklists at network middleboxes to deny

access to bootstrap nodes. For example, IDSes, enterprise firewalls, or ISP routers can drop traffic intended for bootstrap nodes. This approach is very similar to blocking any other malicious IP addresses, and is subject to the usual challenges. Defenders must keep blocklists up-to-date as malware authors update the IPs they connect to. To the advantage of defenders, any time malware authors are forced to update the IP addresses that bootstrap nodes may be found at, they run afoul of the "sunk cost" problem where infected machines that cannot be updated become useless. A similar argument applies if malware chooses to access bootstrap nodes using hard-coded DNS domain names instead of hard-coded IP addresses. Additionally, traditional interventions against domain names apply in that situation as well. Thus, while intervening at bootstrap nodes poses more of a challenge than intervening at centralized proxies, defenders still have viable options to choose from.

5.4.5 Interventions at the database

In traditional DNS, defenders can sinkhole the domain of an authoritative nameserver or seize the server itself to prevent malware accessing a C2 domain record. This intervention is impractical for blockchain names, because instead of a single machine acting as the authoritative nameserver, every blockchain node has a copy of the database. Seizing the database would require either taking down every machine in the blockchain, or executing a successful "51%" attack by taking control of more than half of the computing power in the blockchain. Blockchains are generally highly robust against attacks like these, which makes them unlikely to be the most practical intervention for defenders to attempt. However, small naming-specific blockchains with few participants may be more vulnerable.

Case Study #3 — Namecoin's vulnerability to 51% attacks

Like all of the naming-specific blockchains that we study, the Namecoin blockchain has many fewer participants than blockchains like Ethereum. This makes Namecoin more vulnerable than larger blockchains to a "51%" attack. A 51% attack can be executed when an attacker

controls more than half of the computational power of the blockchain, allowing them to rewrite historical transactions or add invalid ones. Gaining control of more than half of a blockchain's computational power is much easier on blockchains with few participants.

Namecoin has already experienced problems in this area. As of 2014, one mining pool known as "DiscusFish" or "F2Pool" consistently controlled more than 60% of the computational power of Namecoin, and on occasion controlled up to 75% [36]. While we did not find any reports that F2Pool had attacked Namecoin, they had the capability to do so. This vulnerability suggests two potential interventions. First, because Namecoin apparently has few licit users [56], interventions that render the entire naming system inoperable are more feasible than they would be on more popular general-purpose blockchains. Therefore, defenders could attempt to take over the entire blockchain and sinkhole all abused names by seizing control of F2Pool. Second, defenders could potentially apply legal pressure to the operators of F2Pool to coerce them to sinkhole certain specific names. This intervention would rely on defenders' ability to find F2Pool's operators and apply legal pressure in the jurisdiction the operators reside in. We predict that such an intervention would be challenging, but the fact that it appears possible at all contradicts the received wisdom that blockchains cannot be taken over directly.

5.4.6 Interventions after the name record is acquired

If an infected host successfully retrieves its C2 record, that record might take several forms. The three that we observed in existing blockchain naming systems that might be useful to malware were IP addresses, traditional DNS domains, and addresses for distributed storage systems like IPFS and SkyNet. Some naming systems also allow users to store arbitrary text as records, which would let malware operators store nonstandard record types like links to social media posts.

Each of these record types are subject to all of the traditional interventions that have already been described, except one: DS addresses. Distributed storage systems provide a form of "bulletproof" hosting, under the limitation that all hosted content must be static files and not

dynamic websites. Any C2 server implemented entirely on such a system must be a simple file with no dynamic content. Infected hosts that wish to contact a distributed storage system must pass through the same steps shown in Figure 5.3 for accessing a blockchain, which means they are subject to the same interventions. For example, a strain of malware called "IPStorm" has already been discovered using IPFS for its C2 server in the wild. IPStorm connects to IPFS using bootstrap nodes [4, 20], which may be seized or blocked.

Another advantage for defenders is that some distributed storage systems, such as IPFS, do not have redundancy: only a single machine hosts each piece of a file. This raises the possibility of discovering the particular machine responsible for hosting a C2 server and seizing it.

A final possibility for intervening with the name record may be to seize names stored in "hosted" or "custodial" wallets. Some businesses, such as cryptocurrency exchanges, provide custodial wallets for users who wish to let the company handle their blockchain-based assets. This service is designed to make blockchain interaction easier for customers, but as a consequence, the business knows the custodial wallet's private key. If a name is stored in a custodial wallet, the business that runs the wallet could seize it [172]. However, a successful intervention must be difficult for malware operators to evade, and we note that malware operators with good operational practices can simply choose not to use custodial wallets.

5.4.7 Intervening with name modification or purchase

Generally speaking, DNS domains are cheaper, easier to modify, and easier to replace than IP addresses, because each IP address represents a compromised machine while new domains can be purchased inexpensively. Blockchain-based domains on general-purpose chains, such as Bitcoin and Ethereum, change this norm. While resolving a name is free, malware operators must pay *transaction fees* (known as *gas fees* on Ethereum) to register or modify names. These transaction fees can be quite expensive. For example, we found that registering a new name on the Unstoppable Domains service cost nearly $80 in gas fees during a period of high

fees. In contrast, the cost of the name itself was $10. While licit users may wait for fees to be low at times of low network congestion, malware operators may not have that choice if they wish to avoid downtime in their campaign. High transaction costs poses challenges for defenders as well. For example, to combat DNS-based DGAs, defenders have the option of registering every domain the DGA will ever generate. This intervention would be much less practical if each registration was nearly an order of magnitude more expensive.

Naming-specific blockchains, such as Namecoin, Emercoin, and Handshake, present a different set of tradeoffs for defenders and malware operators. These blockchains are created with the sole intention of hosting a naming system. With fewer users and correspondingly less demand, these systems' names are usually much less expensive than names in Ethereum-based systems. This enables malware authors to use fast flux or DGA-based strategies, and also may enable defenders to pre-register domains generated by DGAs.

5.5 Measurements of Name Resolution Queries

Our analysis of the registered names in each blockchain naming system (Section 5.3) indicated that malware is not yet utilizing ENS and Unstoppable Domains. In contrast, recent work on the records stored in Emercoin and Namecoin found that these systems were heavily used by malware as recently as 2020 [56]. However, to test whether malware is still using Namecoin and Emercoin and is not using ENS and Unstoppable Domains, it is necessary to analyze not only which names are *registered* in each system, but also which ones are heavily *used*. This is challenging because name resolutions are not transactions: they are read-only operations that do not leave a record on the blockchain. We cannot directly measure usage of blockchain names, but we observed that a side channel might exist to estimate name usage: "leakage" to the DNS. We predicted that since blockchain names require configuring alternate resolution systems, some requests might "leak" into the DNS when misconfigured machines attempt to resolve them as ordinary DNS domains. These leaked names would be visible at the root DNS servers, but

Table 5.4. Examples of malicious Namecoin and Emercoin domains in the October sample of B-root queries.

Malware	Domain	Lookups	Source
Gandcrab	malwarehunterteam.bit	348	[207]
	politiaromana.bit	341	[207]
	gdcb.bit	316	[207]
	zonealarm.bit	628	[206]
	ransomware.bit	1,039	[206]
CHESSYLITE	leomoon.bit	935	[84]
	lookstat.bit	710	[84]
	sysmonitor.bit	519	[84]
	volstat.bit	455	[84]
	xoonday.bit	573	[84]
Dofoil	vrubl.bit	988	[202]
	levashov.bit	1,059	[202]
	vinik.bit	6,265	[202]
KPOT Stealer	kpotuvorot10.bit	1,951	[127]
	star-fox.bit	351	[143]
Team9 Loader	bestgame.bazar	942	[167]
	forgame.bazar	865	[167]
	zirabuo.bazar	51	[167]
	tallcareful.bazar	146	[167]
	coastdeny.bazar	139	[167]
BazarLoader	acegikbcggin.bazar	546	[22]
	acegilbcggio.bazar	467	[22]
Trojan RTM	stat-counter-[0-9]-[0-9].bit	10,498	[25]
Necurs	jfbbrj3bbbd.bit	1,505	[204]
	qcmbartuop.bit	1,316	[18]

would not be forwarded to any other DNS servers, because the roots would respond that the alt-TLDs do not exist. An observer who could see which names were requested at a DNS root server with alt-TLDs corresponding to blockchain naming systems could get a sense for which names are in use.

We therefore took two samples of the names that were requested at the DNS B-root servers over the course of several days. The first sample consisted of names and how many requests were made for each on October 19, 2021. The second sample consisted of names, numbers of requests, and the ASes the requests were made from. It spanned two weeks in April 2022, from April 16 to April 30.

Another advantage of using B-root as a vantage point was that it let us observe requests for *unregistered* names that might indicate the presence of malware. DGAs work by generating a vast number of names, but only a few are ever registered and functional at any given time. These unregistered names do not, of course, appear in our samples of the registered names in each blockchain naming system. An infected host determines which names are registered by simply attempting to resolve them. If an infected host's queries were leaking into the DNS, we theorized that these queries would be very obvious, because the infected host would always receive NXDOMAIN responses from the root. These responses would cause the infected host to assume it has not found the correct C2 name for today and keep trying new names. The flood of nonexistent names with blockchain-based alt-TLDs would be visible when examining queries arriving at B-root.

5.5.1 Frequently accessed names

We first investigated how many days each name was requested on, and found that the vast majority of names are only requested once, on a single day. There were two notable exceptions: a small group (67) of `.bit` names that received a high volume of requests on every day of the sample, and a large group (39,000) of unique `.bazar` names that were each requested on most or all days of the sample. These names belong to the Namecoin and Emercoin naming systems,

respectively.

We analyzed the group of `.bit` names that were requested on all 14 days of our sample and had more total requests than any name requested on fewer than 14 days. 66 names fit this criteria. We submitted them to VirusTotal and found that only 18 names were not labeled malicious by any engine, while 48 were labeled malicious by at least one.

The `.bazar` names that were requested on most (more than 10) days for our sample fell into two categories. The first contained names that appear to be generated by concatenating four lowercase-letter bigrams consisting of a consonant and a vowel (e.g., `acbaelek.bazar`, `acbaelel.bazar`, `acbaelid.bazar`). These names appear to be generated by the malware BazarLoader [45]. The second category contains 38 names that do not appear to be randomly generated. We uploaded these to VirusTotal and determined that 23 were labeled as malicious by at least one threat intelligence service, three were not indexed by VirusTotal, and the remainder were not labeled as malicious. Six of the names were themed around Australian tourism, of which four were labeled malicious and two were not: these names may also be associated with BazarLoader [3].

We note that the most popular Emercoin and Namecoin names each day were largely known to be associated with malware, which we determined by manually searching the Internet. We present a sample of the most popular malware-related names in Table 5.4. These names were taken from the October sample; the days in April had a similarly high number of malicious names that received high volumes of requests.

5.5.2 Unregistered ENS and Unstoppable Domains names

We observed a large number of names, mostly randomly generated names, with alt-TLDs that are used by ENS and Unstoppable Domains. However, we found that these names are actually unrelated to blockchain naming systems and are likely not part of malware campaigns. We drew this conclusion for two reasons. First, the randomly generated names only had one lookup each, and all of these lookups originated from a single AS (AS15169, Google). This is in

contrast to lookups for randomly generated names in Emercoin and Namecoin that are known to be part of malware campaigns: these requests originate from many different ASes and some names receive many more than just one request. Second, not a single randomly generated domain with an ENS or Unstoppable Domains alt-TLD was registered in a blockchain naming system. If these names had been part of a malware campaign, at least one should have resolved to the address of a C2 server at some point. It is possible that B-root only received failed requests from a single misconfigured machine, but this does not match the behavior we observe for malware campaigns that abuse Namecoin and Emercoin.

We predict that rather than being intended for use in a blockchain naming system, these randomly generated names were leaked from local networks, and were never intended to be resolved by either a blockchain naming system *or* the DNS root. A prior study on root DNS queries found that some networks use non-ICANN TLDs internally, under the assumption that queries for those names will never reach external DNS resolvers. However, these queries frequently leak to external networks [63]. We predict that some internal networks use alt-TLDs that coincidentally overlap with the blockchain naming systems' alt-TLDs. We concluded that these names were unlikely to be part of DGA-based malware campaigns, and were also likely unrelated to blockchain naming systems at all.

5.5.3 Requests for registered names from ENS and Unstoppable Domains

Very few registered names from ENS or Unstoppable Domains leaked to B-root: we observed fewer than 400 unique ENS names per day and fewer than 300 unique daily names from Unstoppable Domains. These names also received few requests per day compared to the names from Namecoin and Emercoin. No name received more than approximately 350 lookups, in contrast with the most popular domains in Namecoin, which received an order of magnitude more requests per day. We submitted every ENS and Unstoppable Domains name that received more than ten daily requests to VirusTotal. None were in VirusTotal's database, in contrast with

names from Emercoin and Namecoin, which were largely present and flagged as malicious.

Each of these findings regarding names that leak to B-root support our conclusion that malware still heavily utilizes older systems like Namecoin and Emercoin, but has not yet adopted new systems like ENS or Unstoppable Domains. We predict that this is due to two factors. First, the monetary cost of creating and modifying names in ENS and Unstoppable Domains is much higher than in naming-specific systems like Namecoin and Emercoin. Second, defenders have apparently not yet been able to exert enough pressure on Namecoin and Emercoin to make these systems unattractive to malware operators, because we still see high malware usage of those systems. We hope that the findings in this work will aid defenders in exerting more effective pressure against malware operators.

5.6 Discussion

Out of the five naming systems we examine, we find none so far that present an entirely intractable problem for defenders. For a naming system to present a threat, it must be both easily usable by malware authors and popular enough that blocking its bootstrap nodes, or blocking access to it entirely, will cause significant collateral damage to licit users. For a system to be widely adopted by licit users, it must have three necessary characteristics.

First, the system's name management must be as easy or easier than name management on traditional DNS domains. Users must not be required to write code themselves to interact with smart contracts, as is currently the case with each of the systems we study if the user does not use a custodial wallet. Users also must not be required to run a blockchain node in order to manage their names, as Handshake currently requires to the best of our knowledge.

Second, the transactions that are required to register and update names must be affordable. Transactions on Ethereum, in our experience, cost anywhere between $60 and $140 during the course of our experiments, although we discovered that we were attempting to make transactions during periods of high network congestion and fees were unusually high. Even transaction fees as

low as ten dollars per transaction are far less affordable than transaction fees on naming-specific chains, which can be as low as a few cents. This dynamic may make ordinary users more likely to embrace naming systems built on naming-specific chains, rather than general-purpose chains. However, general-purpose chains may be better known, and therefore more likely to be trusted by users even if transaction fees are higher than on naming-specific chains. A trade-off may therefore exist between affordability and perceived trustworthiness and name recognition.

Third, licit users are unlikely to embrace any naming system that does not have widespread browser adoption. Browser adoption is hindered by naming systems' lack of coordination, which currently leads to name collisions: for example, the alt-TLDs `.wallet`, `.coin`, and `.x` are currently used by multiple blockchain naming systems. Some newly created ICANN TLDs also collide with Handshake TLDs, such as `.music`. Naming collisions present a barrier to browser adoption because the browser would either have to enforce some sort of precedence for systems that include colliding names, or users would have to choose which naming system to use for each name with collisions. Either option will confuse and frustrate users who are unfamiliar with the concepts of namespaces. So far, only browsers that focus on privacy as one of their primary features have chosen to resolve alternate naming systems, and none have chosen to resolve systems that might collide with either each other or ICANN TLDs. Until browsers can resolve an alternate naming system natively, users are unlikely to adopt that naming system.

We conclude that the higher ease of use of purchasing, modifying, and resolving traditional DNS domains is a very high barrier for blockchain-based naming systems to overcome. As long as blockchain naming systems are not widely adopted, we predict they will not become entirely intractable problems for defenders.

5.7 Related Work

Kalodner et al. performed the first study to our knowledge of Namecoin in 2015 [123]. They conclude that the Namecoin ecosystem was "dysfunctional:" only 28 out of 120,000 registered names were valid, not squatted, and had nontrivial content.

Patsakis et al. present an analysis of potential weaknesses and user risks of Namecoin and Emercoin, including the risks of squatting, 51% attacks, phishing, and abuse by malware [170]. The authors also provide an overview of the names stored in these systems, and found that many names registered in the Alexa Top 1K were also registered under Namecoin and Emercoin's alt-TLDs. Most of these squatted names redirected to pornographic websites.

Casino et al. analyzed the IP addresses in Namecoin and Emercoin records [56]. They first identified malicious IP addresses using several threat intelligence databases, and then clustered all the IPs into "malicious," "suspicious" and "benign" categories with a "poisoning" approach. An IP was labeled "malicious" if a threat intelligence database categorized it as such. It was labeled "suspicious" if it appeared in the same wallet, was resolved to by the same domain, or shared the same email TXT record as a malicious IP, and "benign" if it had no connection to a malicious IP. Casino et al. discovered that only 8% of the IPs in Emercoin and 28% of those in Namecoin had no association with malicious IP addresses. While this paper mentioned the existence of more recent blockchain naming systems, it did not perform an analysis of any system except Namecoin and Emercoin.

Numerous other blockchain-based naming systems have been proposed, including the Blockstack Naming System [36], Bitforest/Conifer [79, 80], BlockDNS [181], and Nebulis [124]. To our knowledge, only Blockstack has evolved into a commercial product. We excluded the Blockstack Naming System from this work because it does not appear to be as popular as the other systems we study.

Other work has analyzed the ways in which blockchain technologies in general might be abused by malware. Pletinckx et al. analyzed Cerber ransomware and found that it used

blockchain wallet addresses as domains [176]. Hassan et al. point out that blockchain nodes reside in so many different legal jurisdictions, it will be difficult for regulators to control what information gets passed across country borders [103]. Moubarak et al. present a theoretical design for malware to store pieces of its payload on Bitcoin [156].

Relatively little work has been done on defenses against malware that uses blockchain naming systems. Huang et al. developed a machine learning-based detection method for distinguishing malicious blockchain-based names from benign names in DNS traffic [112]. Hu et al. presented a brief comparison of DNS and Bitcoin-based naming systems, and noted that small, naming-specific blockchains like Namecoin were vulnerable to 51% attacks [109].

Prior work has evaluated the effectiveness of interventions that target DNS domains. Kesari et al. provide an overview of legal intervention methods and cites their use in a number of malware takedowns [125]. Wang et al. studied the use of TROs to seize storefronts run by spammers [215]. Liu et al. analyzed the effectiveness of two interventions that were initiated by registrars and designed to stop spammers from registering domains [141]. Prior literature has also analyzed interventions based on taking down hosting providers, and concluded that these interventions have modest or mixed effectiveness [51, 131, 164, 38].

5.8 Summary

While decentralized naming and hosting systems pose challenges, they cannot entirely eliminate their reliance on systems with centralized authority. Whenever malware uses a centralized resource to enable its use of decentralized ones, defenders can intervene. Defenders cannot serve legal takedown orders to a centralized registrar to take down a blockchain domain, but they can prevent malware from accessing the blockchain in the first place, or target the DNS domain or IP address that the blockchain domain resolves to. We examined existing blockchain-based naming systems and found that naming systems on general purpose blockchains are not currenly attractive to malware because of their high cost. In contrast, systems on naming-specific

blockchains present an ongoing threat, but these systems are susceptible to defenses such as blocklisting every IP address stored in the name records, blocking the proxies that resolve the names, or blocking the system entirely, because so little licit content exists on those blockchains. We conclude that for a naming system to be truly more dangerous than DNS, it must achieve widespread adoption as well as inexpensive transactions and high ease-of-use, and no existing naming systems have yet achieved all three characteristics.

Chapter 5, in part, is a reprint of the material as it appears in *Proceedings of the APWG Symposium on Electronic Crime Research (eCrime) 2022*. Audrey Randall, Wes Hardaker, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Measuring UID Smuggling in the Wild

The final naming system examined in this dissertation is not a formal, named system like the DNS, Namecoin, or the Ethereum Name Service. Instead, it is the collection of user data, mapped to unique user identifiers, that is maintained by web trackers. By viewing this collection of data as a naming system, we were able to apply the same insights as described in previous chapters: specifically, that participating in the naming system would allow us to measure the prevalence of an adversary's attack. The attack in this case is a privacy-invasive web tracking technique that we label "UID smuggling;" the adversaries are third-party web trackers. Web trackers collect user behavior data and organize it in their own databases by assigning each user a user identifier (UID). Seen through the lens of a naming system, this UID is a name, and its record contains a user's browsing habits and interests. Trackers wish to collect user data from as many websites as possible, to build the most complete profile of a user. A tracker must therefore be able to link any new data it collects for a user to that user's unique UID. Ideally, then, a tracker must maintain a global namespace of UIDs, so that it can know a user's UID no matter which tracker-containing website the user visits.

However, browsers have begun to implement a privacy defense called "partitioned storage" that breaks this global namespace into much smaller per-website namespaces. When patitioned storage is in place, trackers can no longer read or write a global UID for each user: they must in theory assign a new UID to a user for each website the user visits. The goal of this defense is

to prevent trackers from knowing that user data collected from different websites all belongs to the same user, thus making trackers' user profiles much less privacy-invasive. Unfortunately, to evade partitioned storage defenses and rebuild their global namespaces, trackers have begun to use UID smuggling. UID smuggling allows trackers to pass UIDs between websites as a user navigates between them, thus building an approximation of a global namespace by linking identifiers from the per-website partitioned namespaces.

In this chapter, we estimate the prevalence of UID smuggling by acting as participants in trackers' naming systems. By posing as users navigating between websites, we observe tracker's `get` and `set` requests to the naming system. We are thus able to estimate the frequency at which a real user would encounter this attack in the wild. We now present the details of this attack and our method of studying it in more depth.

6.1 Overview

Over the past few years, tensions have deepened between those collecting detailed user behavior data for advertising purposes and privacy-conscious users who do not want to be monitored. While there are some efforts to find a compromise between these positions (e.g., allowing the collection of aggregated, anonymized data [48, 96]), none have yet managed to satisfy advertisers or privacy advocates [70, 187, 213]. In the absence of such a solution, privacy-focused browsers (i.e., browsers for which privacy is seen as a competitive advantage) have rolled out changes that block one of the core mechanisms used by *third-party trackers* to aggregate information about a user *across different websites*, thereby building a profile of that user.

Previously, third-party trackers could build user profiles across websites because information stored by the tracker was accessible to that tracker across all websites that include it. Trackers commonly used third-party cookies for this purpose, although any type of browser storage could be used. Trackers could use this shared storage to build shared state for each user across every website that included the tracker. However, several browsers are now employing

an anti-tracking defense called "partitioned storage," which removes this sharing ability. By partitioning all browser storage by the domain of the top-level website, browsers intended to prevent trackers from linking user information across sites.

However, trackers have responded by implementing a new class of tracking technique that we call *UID smuggling*. UID smuggling allows trackers to share a user's information across websites by modifying the user's navigation requests. The tracker accomplishes this style of tracking by decorating users' navigation requests with identifying information, which will then be shared across first-party boundaries. The tracker may also choose to momentarily redirect the user to its own domain, where it can record this smuggled information as a first party itself. In each case, trackers use UID smuggling to regain the ability to link user identifiers across sites, circumventing the browser's attempt to partition such information.

This work presents the first systematic measurement of UID smuggling in the wild. We make the following contributions:

1. We perform the **first systematic measurement** of UID smuggling in the wild.

2. We construct a multi-stage **analysis pipeline**, nicknamed CrumbCruncher, to crawl the Web and measure how frequently UID smuggling occurs.

3. We improve on prior techniques for **differentiating user identifiers** from other values and **synchronizing multiple crawlers**.

4. We categorize the **behaviors** of trackers, including which categories of sites are more likely to include UID smuggling.

5. We contribute to countermeasures against UID smuggling, both by sharing our hand-edited **dataset**, and by publishing our tool for finding new instances, CrumbCruncher.

The remainder of our work is organized as follows. Section 6.2 covers the background of navigational tracking and related work. Section 6.3 describes the design of our crawler,

CrumbCruncher, and its capabilities and limitations. Section 6.4 discusses ethical ramifications. Section 6.5 presents our findings, including the most common participants in navigational tracking and a summary of their behaviors and categories. Section 6.6 describes the limitations of our work. Section 6.7 details our contribution to the countermeasures that various entities have taken against navigational tracking. Section 6.8 discusses related work, and Section 6.9 concludes.

6.2 Background

Advertisers want to track user activity across sites for a variety of purposes, including performing identity resolution and supporting affiliate marketing, but such capabilities represent a significant threat to user privacy.

For over a decade, browsers allowed advertisers to perform cross-site tracking functions with third-party cookies. But because this capability presents a threat to privacy, several popular browsers have implemented *partitioned storage* to isolate third-party cookies so they cannot be used for cross-site tracking. At the time of writing, three browsers — Firefox, Safari, and Brave [119, 219, 111] — all use partitioned storage by default. Partitioned storage uses a hierarchical namespace, where the hierarchy is based on the domain of the frame that contains the cookie-storing element. Figure 6.1 shows the difference between flat and partitioned storage from a tracker's perspective. When flat storage is in use, the tracker can read from or write to the same storage area regardless of which website it is on, but when partitioned storage is implemented, the tracker accesses a different storage area on each website that loads the tracker. This prevents trackers from assigning the same user-identifying cookie (represented by the gingerbread man icon) to users across sites. A similar system is used for other browser storage locations, such as local storage.

To circumvent the protections that partitioned storage provides, advertisers are increasingly using *UID smuggling*. UID smuggling modifies a user's navigation requests by

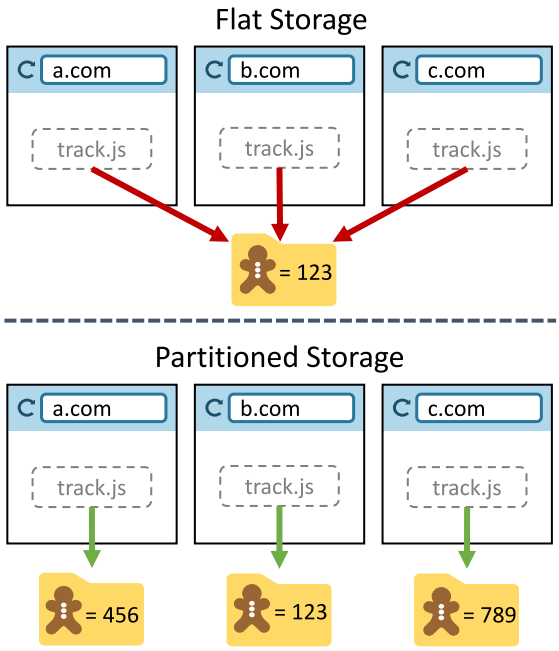

Figure 6.1. Flat storage versus partitioned storage.

adding information to the navigation URLs in the form of query parameters. UID smuggling may also redirect the user to one or more third-party trackers before redirecting to the intended destination.

Figure 6.2 shows this process in detail. In UID smuggling, the user is sent through a **navigation path**. This path begins at the **originator** website, where the user clicks a link (step ①). When the link is clicked, the page itself or a tracker on the page decorates the URL by adding the originator's user identifier (UID) as a query parameter. The link may be either a first party or a third party link, because any script on a webpage may modify any link within its frame. Third party scripts are often loaded within the top level frame of the page, so they may modify any link within the top level frame. After the user clicks the link, the navigation path passes through zero or more **redirectors**, which are invisible to the user but are permitted to store first party cookies (step ②). Each of these redirectors has the ability to store the UID from the query parameter as a cookie or local storage value under the redirector's domain. Finally, the user is sent to the **destination**, the website the link originally pointed to (step ③). The destination may

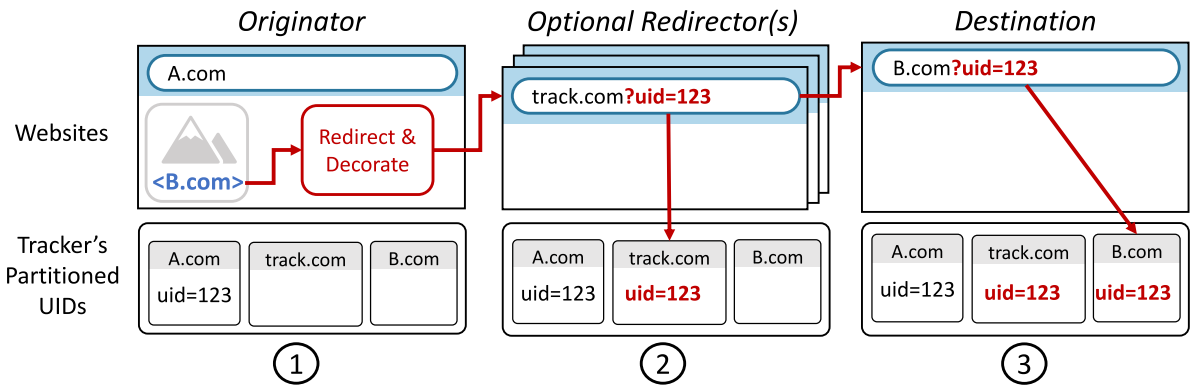

Figure 6.2. How UID smuggling allows trackers to circumvent partitioned storage.

also store the UID under its own domain. If there are no redirectors in the navigation path, the threat to users is simply that the UID gets passed between the originator and the destination. This may occur when the tracker that performs the UID smuggling is confident that its scripts will be present on both the originator and destination page, to decorate the link and collect the UID respectively. Thus, trackers using UID smuggling regain the ability to share UIDs across websites with different domains, in defiance of the browser's partitioned storage protections.

UID smuggling is related to, but more powerful than, two previously studied tracking techniques: bounce tracking and cookie syncing. Bounce tracking also modifies a user's navigation path by redirecting them through tracking sites that can store first-party cookies. Bounce tracking allows a tracker to record which originator and destination websites a user has visited, but not to aggregate any information about a user's behavior on those websites (the links the user clicks, purchases the user makes, etc.). This is the case because no link decoration is used to insert UIDs into the navigation path: if such link decoration is used, the technique becomes UID smuggling. A bounce tracker thus cannot link together the different UIDs it has assigned to a user across different websites. Both UID smuggling and bounce tracking are part of a class of tracking techniques known as "navigational tracking."

Cookie syncing allows multiple third parties on a *single* first-party site to share UIDs with each other. However, if partitioned storage is in place, third parties cannot share information

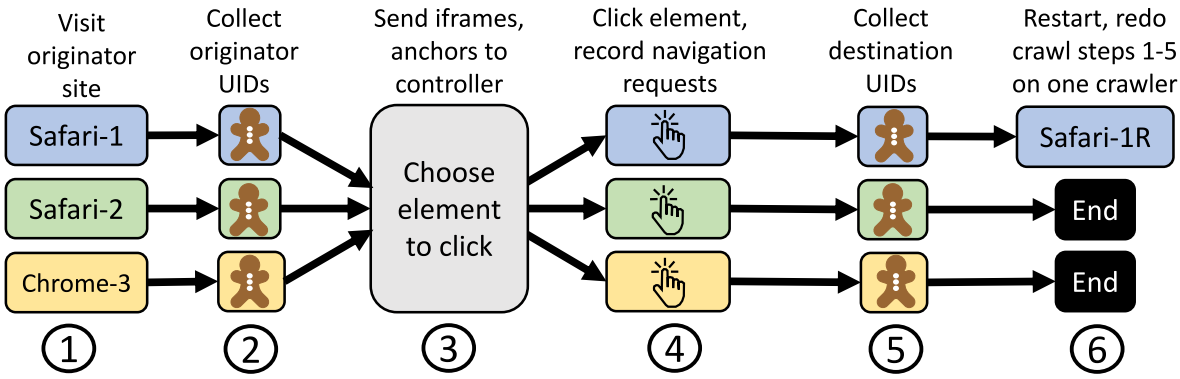

Figure 6.3. A single step of the ten-step random walk that CrumbCruncher performs for each seeder domain.

across first-party websites using cookie syncing. When partitioned storage is in use, the storage available to trackers on the destination site is partitioned away from their storage on the originator. Thus, all trackers on the originator can share their UIDs with each other, and all trackers on the destination can do likewise, but trackers on the originator cannot share UIDs with trackers on the destination.

6.3 Methodology

In this section, we describe CrumbCruncher, a web crawling system based on Puppeteer that measures the prevalence of UID smuggling in the wild. CrumbCruncher's goal is to collect as many potential cases of UID smuggling as possible, and then distinguish the benign cases from true UID smuggling by determining which smuggled values are truly UIDs. To collect potential UID smuggling, CrumbCruncher employs multiple synchronized crawlers that simulate a set of users, with an additional, trailing crawler that simulates a user returning to each site. CrumbCruncher must then identify which potential UIDs are truly UIDs by comparing them across each crawl: values that vary across the set of different users but remain static for the repeat visitor are likely to be UIDs.

The canonical approach for identifying UIDs in prior work is to use only two crawlers to compare potential UID values across users. Unfortunately, these studies have been forced

to discard a large number of potential UIDs from their analyses under two circumstances: first, when the potential UID only appeared on one crawler instead of both, and second, when the potential UID might be a session ID instead. Because we expect UID smuggling to be rare and difficult to find in the wild, we require CrumbCruncher to discard as few UIDs as possible. CrumbCruncher achieves this goal in three ways. First, it distinguishes UIDs from session IDs more accurately than prior studies, which allows it to retain UIDs that would have been discarded by previous common strategies [132, 86, 85, 29]. Second, when potential UID smuggling does not appear on all crawlers, CrumbCruncher applies programmatic and manual heuristics to identify UIDs, rather than discarding the cases entirely as prior work does [93, 86, 85, 132, 29]. Finally, CrumbCruncher introduces a novel method for synchronizing web crawlers that click iframes, which allows it to collect data from the elements that are most likely to contain UID smuggling. CrumbCruncher also uses four synchronized crawlers, rather than two, giving it multiple chances to observe each potential UID across two crawlers. Each of these improvements allows CrumbCruncher to collect or retain more data than previous systems.[1]

6.3.1 Crawling the Web

CrumbCruncher collects a sample of websites that contain UID smuggling by performing ten-step random walks. Each random walk begins at a "seeder domain" taken from the Tranco list of the globally most-popular 10,000 domains [137].[2] Each of CrumbCruncher's multiple crawlers follows the same walk.

At each step of a walk, CrumbCruncher records all first-party cookies, local storage values, and web requests on the originator page. Next, it chooses either a frame (<iframe>) or anchor (<a>) element to click on, in an attempt to trigger navigation. CrumbCruncher selects iframes because we expect them to contain advertisements which might use UID smuggling. CrumbCruncher also follows anchors because many webpages do not contain iframe elements.

[1] For more details on how prior work identified UIDs, please see Section 6.8.

[2] We choose 10,000 seeder domains because several prior studies also used that number [93, 76, 182].

Regardless of element type, CrumbCruncher preferentially chooses elements that navigate to a URL with a different registered domain than the current page. If such an element does not exist, CrumbCruncher selects one at random. For each click that triggers a navigation, CrumbCruncher's browser extension collects all navigation web requests by implementing a handler for the `chrome.webRequest.onBeforeRequest` event. Upon arriving at the destination page, CrumbCruncher again records all first-party cookies, local storage values, and web requests for ten seconds.

CrumbCruncher repeats this navigation process, starting at each new page loaded by the click in the previous step, nine times. It then selects a new seeder domain to start the next random walk. CrumbCruncher retains browser state (including cookies and storage values) for the duration of each walk and discards it when beginning a new walk. CrumbCruncher proceeds in this depth-first manner to maximize the number of distinct pages visited, rather than maximizing the elements visited per page, to distribute the sites it performs clicks on as widely as possible. This strategy minimizes CrumbCruncher's potential impact on advertisers (see Section 6.4 for more details). It also allows CrumbCruncher to observe websites that range in popularity, rather than staying within the ecosystem of popular websites.

6.3.2 Detecting potential UID smuggling

The goal of CrumbCruncher's is to identify cases where a UID has been smuggled — i.e., passed across sites in defiance of browser protections — which requires differentiating UIDs from non-tracking tokens. We use the term "token" to refer to any potential UID found inside the value of a name-value pair, whether that pair is a first-party cookie, a local storage object, or a query parameter. CrumbCruncher builds on prior work that identifies UIDs by comparing the tokens that are passed by two different users access a particular website [29, 86, 85, 132, 93, 208]. However, instead of using two crawlers, CrumbCruncher uses four.

Three of the four crawlers — named Safari-1, Safari-2, and Chrome-3 — each simulate a different user on a Safari or Chrome browser. These three crawlers, which run in parallel, allow

CrumbCruncher to discard tokens that are the same across users and are thus unlikely to be UIDs. We explain how CrumbCruncher spoofs browsers and why we simulate both Safari and Chrome in Section 6.3.4 and how we impersonate different users in Section 6.3.5. The fourth crawler, Safari-1R, simulates the same user as Safari-1. Safari-1R checks whether the same token is observed when a webpage is accessed twice by the same user: specifically, Safari-1R repeats each crawl step immediately after Safari-1 finishes it. Safari-1R allows CrumbCruncher to discard tokens that differ when observed repeatedly by the same user, and thus are probably session IDs, not UIDs; Section 6.3.7 provides more details on this process.

6.3.3 Synchronizing multiple crawlers

One underlying assumption behind the multi-crawler methodology is that all browsers are accessing the same version of a particular webpage: the four crawlers must visit the same URL and click the same elements on each page. Figure 6.3 illustrates this process. However, we find that keeping the crawlers synchronized presents a significant challenge due to the dynamic nature of the web. Determining which elements are the same on different instances of the same webpage is not straightforward. Even when accessed in parallel, websites often load dynamic content: elements that appear on one crawler's page might not appear on the others'. We also find that even when elements are the same (e.g., iframes that load the same content), they might not appear in the same locations or with the same size. Additionally, CrumbCruncher clicks iframe elements, which often do not have any attribute that identifies where a user will navigate when they click the iframe. Determining which iframe elements are equivalent across different instances of a webpage is more challenging than comparing anchor elements, which almost always have an easily comparable <href> attribute.

To mitigate this issue, CrumbCruncher uses a central controller (a local HTTP server) to choose the element that Safari-1, Safari-2 and Chrome-3 click in parallel. Upon loading a page, each crawler sends a list of all anchor and iframe elements on that page to the central controller. These lists contain the elements' properties, location, bounding boxes, and x-paths.

The controller compares the three lists to find elements that are the same across all three instances of the page. We consider elements to be the same if any of three heuristics are met:

1. They are anchors and their `href` values are the same (not including query parameters).
2. They have the same HTML attribute names (the values may differ) and similar bounding boxes (the *y*-coordinate may differ, to allow for elements that render at different heights on the page).
3. They have the same HTML attribute names and x-path.

These heuristics are imperfect: they may incorrectly label elements as the same when they are not, or incorrectly discard elements. To mitigate these possibilities, CrumbCruncher compares the fully qualified domain name (FQDN) of the site each crawler has landed on at the end of every crawl step. If all three FQDNs are not the same, CrumbCruncher terminates the walk. We still include data from this unsynchronized step in our analyses, because this situation often occurs when CrumbCruncher has clicked on different advertisements that each exhibit a separate instance of UID smuggling.

We evaluate the effectiveness of these heuristics and find 7.6% of all crawl steps fail because CrumbCruncher is unable to find an element that is the same across all three synchronized crawls: this type of failure occurs at step ③ in Figure 6.3. A further 1.8% of crawl steps fail at step ⑥ because the clicked elements were not actually the same, and led to different destination websites. The only other significant reason why a crawl step might fail is if CrumbCruncher fails to connect to the website because of a network error (`ECONNREFUSED`, `ECONNRESET`, etc.) at step ①, which occurred on 3.3% of the sites it attempted to visit. We expect the probability of any of these failures occurring to be independent of the step of the random walk CrumbCruncher was on.

6.3.4 Impersonating different browsers

All four crawlers use Chrome (version 95 or 92) because our chosen crawling framework, Puppeteer, is designed for that browser. However, CrumbCruncher impersonates Safari on three of our four crawlers by spoofing the `User-Agent` string.[3] We chose to test Safari and Chrome because at the time of writing, Safari implemented partitioned storage by default, while Chrome's own defenses against third-party cookies were optional (we enabled them for our study). Our hypothesis was that trackers might use UID smuggling more frequently on Safari to evade its ubiquitous partitioned storage protections. Our Chrome-3 crawler was originally intended to test this hypothesis, but we were unable to use it for this purpose: UID smuggling cases quite often appeared on only one crawler, regardless of whether that crawler was one of the three Safari crawlers or the Chrome crawler (see Section 6.3.7). Differentiating cases where content that performed UID smuggling was loaded dynamically from cases where UID smuggling occurred deliberately on Safari and not Chrome proved to be impossible, so we simply use Chrome-3 as another distinct user to identify UIDs.

We note that while spoofing the `User-Agent` string does change the value of `window.navigator`, which is commonly used as a proxy for identifying the browser, it is not a foolproof method of impersonating a browser. Websites may use more sophisticated methods to identify a browser, such as comparing the codecs it supports [210]. However, we do not believe this is a significant problem for our study, because relatively few websites go to such lengths: Vastel et al. crawled the Alexa top 10,000 websites and found that only 93 appeared to use sophisticated fingerprinting techniques to identify the browser that was loading them [210]. We therefore consider the risk of sites misidentifying our browser to be small, given how few websites appear to use fingerprinting to identify browsers.

[3] We use the Safari `User-Agent` string `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Safari/605.1.15`.

6.3.5 Impersonating different users

The Chrome browser differentiates users by storing profiles in a folder called the "user data directory" [183]. To simulate a new user at the start of each random walk, each crawler starts with a new user data directory. This folder is modified from the default in two ways: first, third-party cookies are disabled, and second, a Chrome extension is installed that records web requests.

One potential limitation of our user simulation method is that websites may generate UIDs using fingerprinting, i.e., by examining factors like `User-Agent` string, supported fonts, hardware, and more.[4] Many of these inputs are identical across all four crawlers since they are run on one machine. If a tracker generated its UIDs using fingerprinting, assigned the same UID across multiple crawlers, and then performed UID smuggling, CrumbCruncher would erroneously discard those cases. However, we find that this rarely occurs by performing the following experiment.

We observe that CrumbCruncher will not discard potential instances of UID smuggling that only appear on a single crawler: only instances that appear on multiple crawlers and have identical UIDs will be discarded. If CrumbCruncher is erroneously discarding instances of UID smuggling, we would expect to see very few cases that both occur on multiple crawlers *and* originate on sites that perform fingerprinting.

To test this hypothesis, we separate cases of UID smuggling into two groups: the cases that originate on sites that are known to employ fingerprinting, and cases that originate on other sites. To determine which sites use fingerprinting, we use the list of fingerprinters found by Iqbal et al. [117]. Only 13% of UID smuggling in our data originates on pages hosted by one of Iqbal et al.'s fingerprinters. We then divide both groups again, into the instances that occur on a single crawler and the instances that occur on multiple crawlers. Next, we compare the proportion of single-crawler to multiple-crawler instances in the fingerprinting group to the non-fingerprinting

[4] IP address is generally too variable to be used as an input by fingerprinters [82].

group. In the fingerprinting group, 44% of UID smuggling cases occur on multiple crawlers, whereas in the non-fingerprinting group, 52% of cases occur on multiple crawlers. While the two-proportion Z test suggests that this difference is statistically significant — and, therefore, that CrumbCruncher likely missed some cases of UID smuggling due to fingerprinting — the difference is small. The relative difference between populations suggests CrumbCruncher may have missed on the order of 13 cases of UID smuggling on sites that employ fingerprinting.

6.3.6 Identifying potential UID smuggling

Once CrumbCruncher has finished collecting data, we search for potential UID tokens that were transferred across first-party contexts. We define "different first-party contexts" as the case when the site the token was originally found on has a different registered domain than any of the sites that eventually received the token, whether those sites are redirectors or the ultimate destination.

We extract potential UID tokens from cookies, local storage, and query parameters by recursively attempting to parse the value of each name-value pair[5] as JSON or URL-encoded values. For example, if a query parameter contains a JSON string that itself contains several URL-encoded tokens, we extract each URL-encoded token individually.

We then discard all of the tokens that were not passed across at least one first-party context as a query parameter. For example, if the same token was found on both the originator site and the destination, but was not passed from the originator to the destination as a query parameter, we discard it. We find that the vast majority of these particular tokens are not used in UID smuggling, but rather false positives that happen to appear on both websites, such as location or language specifiers.

However, we keep tokens that only get passed across part of a navigation path. For example, if a token appears as a query parameter in the URL of a redirector, then gets passed

[5]We do not look for tokens in the names of name-value pairs because Fouad et al. found that storing UIDs as names rather than values was a very uncommon practice [93].

to the destination, we keep it even if it did not appear on the original URL of that navigation path. We note that while we cannot tell whether a site that received a UID did anything harmful with it, the fact that the site received the UID at all represents a privacy risk. Tokens are also not required to appear as cookies or local storage values: they can appear on the originator and destination as query parameters in third-party web requests.

6.3.7 Identifying UIDs

After collecting all potential cases of UID smuggling, we identify and discard all of the cases that transfer harmless values rather than UIDs. Examples of harmless values include timestamps, language specifiers, session IDs, and so on. While performing this analysis, we discovered that cases of potential UID smuggling fell into two categories: we labeled these categories "static" and "dynamic." Static UID smuggling occurs on elements that are always the same on every visit to the page. Consequently, cases of static UID smuggling appear on all four crawlers. Dynamic UID smuggling occurs on elements that load different content on different page visits. Cases of dynamic UID smuggling appear on fewer than all four crawlers, despite our efforts to keep the crawlers synchronized. Identifying UIDs in static UID smuggling is simpler than in dynamic UID smuggling, and we describe our procedure for the static case first.

Identifying UIDs in the static case

To track an individual user, a UID must be the same across all website visits by the same user and different across visits to the same website by different users. Consequently, we discard any token that is the same across our crawlers that simulate different users, since these cannot be UIDs. However, it is also necessary to discard tokens that differ across a single user, since these tokens are likely to be session IDs that are not used for user tracking. Prior work discarded session IDs by discarding all tokens whose lifetime was less than a specific time, such as 90 days [132, 86, 85] or a month [29]. CrumbCruncher improves on prior work by comparing potential session IDs across Safari-1 and Safari-1R, which simulate the same user visiting the

Table 6.1. Crawler combinations where UIDs appeared.

User Profiles	# Tokens
2 identical plus 1 or more different profiles	325
2 or more different profiles only	171
2 identical profiles only	20
1 profile only	445

same website twice, and discarding the tokens that differ across these crawlers. A sampling of data collected from one of our crawler machines indicates that 16% of the UIDs we identify have a lifetime of less than 90 days, and 9% have a lifetime shorter than a month. These UIDs would have been missed by prior work that uses lifetime to determine whether a token is a session ID.

Identifying UIDs in the dynamic case

Unfortunately, we found that the majority of potential UID smuggling instances were dynamic and thus did not occur on all four crawlers: in fact, many instances occurred on only a single crawler. For example, we encountered many cases where each crawler loaded the same originator website and clicked the same iframe element, but the iframes contained different advertisements, so each advertisement presented a different navigation path and arrived at a different destination. We classify tokens that appear on fewer than four crawlers in the following manner:

1. If a token is present in any two crawls with different user profiles, and its value is the *same* across those crawls, we discard it.

2. If a token's name is present in Safari-1 and Safari-1R, which have the same user profile, and its value *differs*, we discard the token.

We are left with two classes of tokens: tokens that are present in only a single crawl, and tokens that only appear in crawls with different profiles (and have different values across each crawler). For these tokens, we employ both the programmatic heuristics used by previous work and manual sorting.

We base our programmatic heuristics on those of prior studies [86, 85, 132]. We remove tokens that appear to be dates or timestamps, tokens that appear to be URLs, and tokens that are less than eight characters long. We do not impose any restrictions based on cookie expirations. However, even after we applied these filters, manual inspection of the remaining tokens revealed a high number of obvious false positives. These included natural language strings separated by delimiters (such as "Dental_internal_whitepaper_topic," "share_button"), concatenated words with no delimiter ("sweetmagnolias," "trustpilot"), semi-abbreviated words ("navimail"), acronyms ("en-US"), and more. Filtering most of these out programmatically presented a significant challenge.

We therefore concluded that programmatic heuristics would be insufficient to distinguish UIDs from other tokens, and resorted to removing obvious false positives by hand. Our final, conservative strategy is to remove tokens that are composed of any combination of natural language words, coordinates, domains, or obvious acronyms like "en-US." Table 6.1 shows how many of the final set of UIDs were present on various combinations of crawlers.

In the end, we manually removed 577 out of 1,581 tokens because our programmatic filters failed to recognize them as non-UIDs. This number is significantly higher than we expected and underscores the value of attempting to observe UIDs across as many crawlers as possible.

6.3.8 Implementation

We implemented CrumbCruncher using both Puppeteer, to automate site visits and record cookies and local storage, and a custom Chrome extension, to record web requests. We use Puppeteer in "headful" mode, using the monitor emulator XVFB [223], to reduce the chance that CrumbCruncher will be identified as a bot. While Puppeteer is capable of recording most web requests, it cannot guarantee that it can attach request handlers before any requests on a page have been sent [44, 47]. We found during initial testing that this led to a significant number of missed requests; hence, CrumbCruncher records requests using a browser extension instead. CrumbCruncher runs on twelve Amazon EC2 `t2.large` instances. Each EC2 instance has a

different set of 834 seeder domains. The full crawl of 10,000 seeder domains takes approximately three days to complete.

6.4 Ethics

CrumbCruncher's mechanical measurements do not reflect the interests or intentions of individual consumers. Because CrumbCruncher cannot be influenced to make a purchase after clicking an ad, there exists a view that our measurements represent potential economic harms against the profits of the advertising industry and its clients, and that such economic harms may represent ethical considerations.

Unfortunately, it is difficult to precisely quantify this potential economic harm for a variety of reasons. First, different ad placements can have different payment triggers, such as cost-per-mille (CPM), cost-per-click (CPC), or cost-per-action (CPA). Second, different advertisements are priced differently for each publisher. For example, CrumbCruncher predominantly engages with display ads, which are commonly placed via real-time auctions. The prices of these ads fluctuates continuously based on a variety of factors, including conversion rates. Finally, advertising platforms commonly refund expenditures on ad clicks that they deem to be "bots." To establish an *upper bound* on the economic impact of our study, we estimate that our study involved fewer than 50,000 ad clicks. If we assume that all of these ads were placed on a top-tier network (e.g., Google Display Ads, with average CPC of $0.67 and average CPM of $3.12 [203]), and that none of our clicks were identified as bots, the total cost would be somewhere between $152 (all CPM ads) and $33,000 (all CPC ads). This expenditure in turn would be spread across the range of advertising networks and advertisers found in our random walks (a number that is also hard to estimate, but likely represents an average cost of a dollar or less for each).

Based on this assessment, we argue that the actual costs borne by the advertising ecosystem due to our experiments are modest. However, even if they were not, we would still argue that measurements such as those described in this paper are ethical and should

Table 6.2. Summary of the navigation paths and their participants measured by CrumbCruncher.

Unique URL Paths	10,814
Unique URL Paths w/ UID Smuggling	850
Unique Domain Paths w/ UID smuggling	321
Unique Redirectors	214
Dedicated Smugglers	27
Multi-Purpose Smugglers	187
Unique Originators	265
Unique Destinations	224

continue to reflect a norm of research practice (as they have for over a decade). The digital advertising market today stands at close to $400B [199] in annual revenue and increasingly profits based on its ability to target ads based on detailed profiles of each user. The incentives to provide ever better targeting are enormous and there is scant evidence that the advertising industry, on its own, is likely to limit such targeting for the benefit of those users who prefer to maintain greater privacy. Thus, one of the only mechanisms for monitoring the evolution in advertising targeting technology — including techniques such as UID smuggling which are designed to bypass privacy protections — is to have researchers engage directly with the advertising ecosystem and measure it. Such efforts are critical to inform consumers, regulators and those technology developers providing improved privacy protections. We believe that such benefits vastly outweigh any modest losses to advertisers.

6.5 Results

We consider two forms of navigation paths in our evaluation. "URL paths" consist of the full URLs of the originator, any redirectors, and the destination (e.g., `a.com/x/y?UID=0` → `b.com/x/y?UID=0`). Domain paths consist only of the domains at each step of the path (e.g., `a.com` → `b.com`).

In total, we observed 10,814 unique URL paths in the data set we gathered using CrumbCruncher. We consider unique URL paths, rather than total URL paths including duplicate

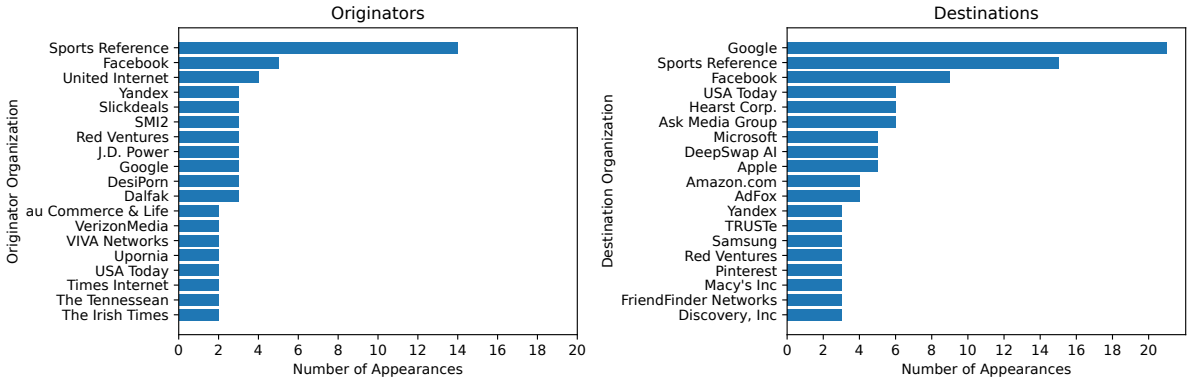

Figure 6.4. Most common entities involved in UID smuggling as originators or destinations.

paths, because this metric gives a better estimate of how many websites participate in UID smuggling.

Using our method for identifying UIDs, we found UID smuggling on 8.11% of the unique URL paths taken by CrumbCruncher. It is interesting that such a non-trivial percentage of advertisers have implemented UID smuggling, especially given that Chrome — the most widely used web browser — still permits tracking with third-party cookies by default.

We speculate that the affiliate advertising market may be driving the adoption of UID smuggling. An affiliate advertising model is one where a company that wishes to publish advertisements, such as a retailer, hires "affiliates" to distribute the advertisements on their behalf. The retailer runs an "affiliate program" which creates the advertisements and gives them to affiliates to distribute. The affiliates earn a commission on every clicked ad that leads to a purchase, known as a "conversion" [139]. Affiliate programs have reportedly been failing to attribute conversions because of browsers' third party cookie blocking [102], and link decoration allows conversions to be attributed correctly.

In the rest of this section we examine the UID smuggling we discovered in detail to understand who is implementing it, how they implement it, and why they implement it.

6.5.1 Redirectors

We start by identifying the trackers involved as redirectors in the navigation paths that include UID smuggling. Again, a redirector is an entity that lies in the middle of a navigation path between the originator and the destination. A smuggler can be any entity along the path that sends or receives a UID, including a redirector, an originator, or a destination. We use unique domain paths instead of URL paths for this analysis, because this metric better captures how widely a redirector is spread across the web, without over-counting repeated instances of UID smuggling by the same entity.

We classify redirectors into two groups: "dedicated smugglers" and "multi-purpose smugglers." We use a conservative heuristic to identify dedicated smugglers that appear to have no purpose in the navigation path besides UID smuggling.[6] Dedicated smugglers exist because they have the right to set first-party cookies, since they are visited in a first-party context, even when partitioned storage is in place. They provide an easy way for trackers to aggregate all the UIDs they collect from different sites into a first-party storage bucket. We consider a redirector a dedicated smuggler if it meets three requirements:

- The redirector appears in navigation paths whose originators have multiple different registered domains,
- The redirector appears in navigation paths that end in destinations with multiple registered domain names,
- The redirector's FQDN is never observed as an originator or destination.

If a redirector does not meet this criteria, we classify it as a multi-purpose smuggler. We separate out dedicated smugglers because we are confident that these domains have no purpose besides UID smuggling, and their sole intent is therefore likely to be enabling trackers

[6] For example, site-specific redirection services (e.g., Twitter's `t.co`) are not considered dedicated smugglers using this classification.

to aggregate users' information across websites. We also predicted that dedicated smugglers might be particularly underrepresented in filter lists that block trackers, because UID smuggling is such a recent technique. Indeed, when we compared the dedicated smugglers that we found to the Disconnect list of trackers [116], 41% of them (11 out of 27) were not yet present in the list.

However, our heuristic is conservative. The less often CrumbCruncher sees a redirector, the less likely it is to observe multiple originators and destinations for that redirector, in which case the redirector would not be classified as a dedicated smuggler. Consequently, some dedicated smugglers might appear in the "multi-purpose smugglers" category.

Table 6.3 shows the 30 most commonly-occurring redirectors in the navigation paths we measured. From this list, 16 of the 30 most common redirectors are dedicated smugglers and 14 are multi-purpose (the multi-purpose smugglers are marked with an asterisk). Of the 16 dedicated smugglers, 14 are owned by advertisers, while the other two (`btds.zog.link` and `secure.jbs.elsevierhealth.com`) have unclear owners or purposes. The most commonly used dedicated smuggler is DoubleClick, which appears in more than 20% of all cases of UID smuggling.

The multi-purpose trackers appear to fill a variety of roles: while all of them perform UID smuggling, some have a separate purpose as well. Some redirect to sign-in pages (e.g., `signin.lexisnexis.com`), host user-facing websites (e.g., `www.facebook.com`), redirect to the English-language version of a site by appending "/en/" (e.g., `www.getfeedback.com`), or upgrade or downgrade HTTP/HTTPS connections (e.g., `kuwosm.world.tmall.com`). Some multi-purpose smugglers are owned by advertising companies, just as the dedicated smugglers are. Two redirectors, `swallowcrockerybless.com` and `d.agkn.com`, appear to be associated with Potentially Unwanted Programs (PUPs) such as adware.

6.5.2 Originators and destinations

Next, we identify the organizations that acted as originators or destinations during UID smuggling. We began with the Disconnect entity list [115], which recorded an owning

organization for 45 out of the 436 unique registered domains of the originators and destinations. We then identified the owners of a further 235 registered domains manually (all of the domains that appeared multiple times, plus as many of the long tail as we could). To manually attribute a hostname to an organization, we use a combination of WHOIS records, copyright ownership information published by the company, and visiting the hostname in a browser. We found that WHOIS was not a reliable method for finding this information as many websites use WHOIS privacy services, so we relied more frequently on copyright information and other publicly available information found via searching the Web. An organization is counted once per unique domain path: if multiple domains owned by a single organization appear more than once in a domain path, the owning organization is only counted once for that path.

Figure 6.4 shows the originators and destinations observed most frequently in our measurements. We present the entities as organizations rather than hostnames because some organizations own multiple hostnames that appeared in our results. We note that many originators might be expected to publish affiliate advertisements, such as sports websites, news organizations, and adult websites, while many destinations might have affiliate advertising programs, such as retailers or technology companies. While we cannot guarantee that these entities participate in UID smuggling as part of affiliate advertising campaigns, many of these organizations are the types of organizations that usually participate in affiliate advertising programs as affiliates and advertisers.

Figure 6.4 also illustrates one particular case of UID smuggling between unexpected organizations. One of the most common cases of UID smuggling in our measurements was a navigation path that led from the originator `instagram.com`, owned by Facebook, to the Google Play Store. This path existed because the button on `instagram.com` advertising the Instagram mobile app always appended `instagram.com`'s UID cookie to the navigation request for `play.google.com`. We were surprised to see that two large advertising companies, that might be expected to be competitors, were apparently sharing UIDs with each other.

Figure 6.4 also contains an example of UID smuggling that was not initiated by an

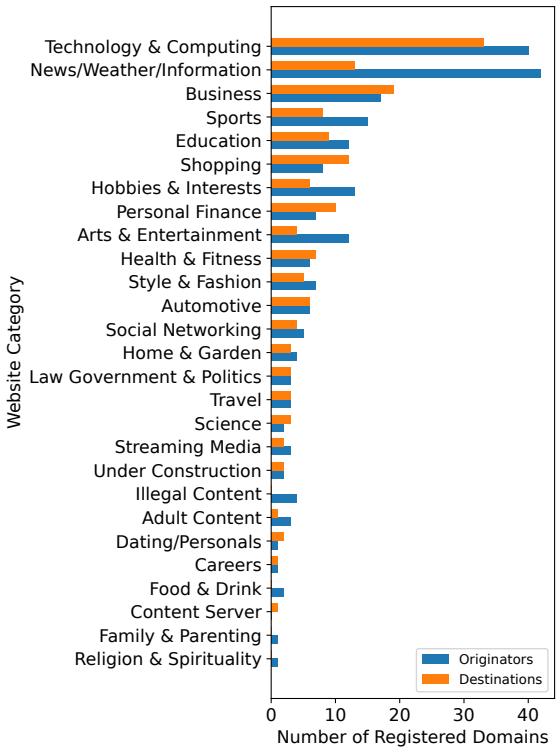

Figure 6.5. Categories of websites that participate in UID smuggling as originators or destinations.

advertiser, but rather used to synchronize information between multiple domains owned by a single company. The most common originator in Figure 6.4 is Sports Reference, an organization that maintains several websites with statistics for popular American sports. This company owns several sports-themed domains whose websites link frequently to each other, such as `hockey-reference.com`, `stathead.com`, `baseball-reference.com`, and others [144]. CrumbCruncher spent several random walks in this ecosystem of websites. We hypothesize that rather than using UID smuggling for advertising, Sports Reference uses it to share information between its own affiliated sites.

Content categories

We further break down the originators and destinations by categorizing them by the topic of their site content. We use the categorization defined by the IAB Tech Lab Content

Taxonomy [113] as provided by Webshrinker [221], whose data set contains 404 domain categories [220]. Out of 339 unique registered domains, 307 had a useful category and 32 were categorized as unknown.

Figure 6.5 shows the most common categories of websites that participate in UID smuggling in our dataset. The counts of websites per category reflect the number of unique registered domains in that category, so that each registered domain is represented only once even if CrumbCruncher encountered it multiple times. For example, even though Facebook's domains are common originators as seen in Figure 6.4, they only appear twice as originators in Figure 6.5: once for `facebook.com` and once for `instagram.com`, both in the "Social Networking" category.

Notably, "News/Weather/Information" is the most common category for originators, and the second most common category overall. This result is consistent with previous studies that found news websites to have an above-average amount of more traditional tracking mechanisms, such as fingerprinting and tracking pixels [85, 117]. Our impression, based on manual inspection of a few of these originators, is that news websites have an above-average number of advertisements in iframes that perform UID smuggling when they are clicked.

Third parties

After a UID has been transferred through the entire navigation path, it may not have finished its journey: third parties on the destination site may also send the UID back to their own servers. Figure 6.6 shows the 20 most common registered domains of the targets of web requests sent from destination sites that included UIDs.

The third-party trackers listed in this figure include trackers that did not appear to use UID smuggling. We note that many requests to third party trackers passed the UID only because the request included the entire URL of the destination site, suggesting that the UID may have been "leaked" to these entities accidentally. This unintended consequence of UID smuggling may present a further privacy harm, in that trackers that do not participate in UID smuggling are

nevertheless gaining access to UIDs that they would otherwise be unable to observe.

6.5.3 Navigation paths

In this section, we examine the characteristics of navigation paths used for UID smuggling, including the features that differentiate them from benign navigation paths.

Figure 6.7 shows the number of redirectors in the middle of each URL path that was used for UID smuggling. The first bar, with zero redirectors, shows the cases where a UID was transferred directly from the originator to the destination without passing through any redirectors in between.

The higher the number of redirectors in a path, the greater the proportion of those paths that contain dedicated smugglers, and the greater the number of dedicated smugglers in each path. We conclude that shorter navigation paths are more likely to have a benign purpose, whereas longer navigation paths are more likely to be used for UID smuggling.

Long navigation paths give multiple trackers the ability to share UIDs with each other. For example, one navigation path started at a coupon-collecting website (`couponfollow.com`), passed through a partner site owned by the same entity, then passed through four different trackers before arriving at the final destination (a retailer). Each of these trackers had the ability to record information about the ad the user had clicked and their apparent interest in the retailer's products.

Long navigation paths can also allow a single tracker to coordinate multiple domains that it controls. If those domains are connected to separate infrastructure (as might be the case if one advertising company acquired another and inherited the acquisition's infrastructure), the company might wish to synchronize the UIDs stored as first party cookies by redirectors. For example, the most common pair of redirectors we observed (where the first domain in the pair immediately redirects to the second domain) is `awin1.com` $\rightarrow$ `zenaps.com`. Both domains are owned by the advertiser AWIN.

UIDs do not always begin at the originator and pass through each redirector before

arriving at the destination: they may appear at any step of the path and cease their journey at any number of hops further along. Each navigation path can also contain multiple UIDs. Figure 6.8 shows how many UIDs traverse each portion of the navigation path. We divide the UIDs that traverse each partial path into two groups: the UIDs that passed through a dedicated smuggler, and the UIDs that passed through either multi-purpose smugglers only or no redirectors at all. For example, the second bar, "Originator to Destination," shows the number of UIDs that were passed through navigation paths with no redirectors.

We observe that the majority of UIDs are transferred across the entire path from the originator, through any redirectors if they exist, to the destination. A tracker might wish to do this when it is reasonably confident that the destination will include one of its scripts, which is capable of storing the UID under the destination's domain. If a tracker is present on the originator and capable of initiating UID smuggling, but is not confident that the destination will contain one of its scripts, it might choose to transfer the UID through only part of the navigation path. These "partial transfer" cases involve a higher proportion of dedicated smugglers, which is further confirmation that the redirectors we label "dedicated" have no other purpose in the navigation path than UID smuggling. We hypothesize that trackers who only send a UID through a part of the navigation path might be less widely used, since they are apparently not confident that the destination will contain one of their scripts.

6.6 Limitations

CrumbCruncher has several limitations. First, we only look for UIDs that are transferred in the query parameters of URLs, and not by other methods. For example, trackers reportedly sometimes decorate the link in the document.referrer header with the UID, instead of the link to the destination page [224]. Our initial reasoning was that there are a wide variety of ways to transfer UIDs, so we could simply check once a crawl was complete for UIDs that had mysteriously appeared on different websites without being passed through a URL. In

practice, this turned out to be difficult: dynamic instances of UID smuggling had to be detected using heuristics, which gave large numbers of false positives when used without the additional information provided by multiple crawlers. It turned out that when the same value appeared on two different websites, the most common reason was that the value was not a UID and had simply happened to be generated on both sites. To reduce our false positive rate and therefore the number of identifiers we had to remove by hand, we chose to consider only values that we had observed get transferred across at least two first party contexts.

Second, if a website uses browser fingerprinting to generate UIDs, our methodology may not fool the site into believing that our crawlers represent different users. As detailed in Section 6.3.5, the effect of browser fingerprinting on our results is very small.

Third, if a tracker uses fingerprinting to identify the browser loading the site, as opposed to the user, it may be able to tell that CrumbCruncher uses Chrome, not Safari. If a tracker only uses UID smuggling on Safari, it may choose not to perform it and CrumbCruncher may miss cases. We expect this to be very uncommon (see Section 6.3.4).

Fourth, our proposed solution to UID smuggling is to strip out the query parameters that contain UIDs. This may cause pages to break, especially in cases where the UID in the URL is used for a benign purpose, such as in a login page. Some login pages send UIDs to the server to determine if a user is already signed in. To test this limitation, we selected ten login pages from our dataset that CrumbCruncher had classified as performing UID smuggling. We manually removed the query parameter that contained the UID from the URL, reloaded the page, and evaluated whether the page changed or broke. We found that seven of the ten sites showed no change. One showed minor visual changes: the <body> element of the page moved down by 20 pixels. The final two pages showed more significant changes: one failed to auto-fill a field in a form and the other took the user to a homepage rather than to a specific subpage. These breakages are a limitation of our proposed mitigation.

Another minor limitation is that the Tranco list of websites includes non-user-facing websites [188]; however, we note that we only failed to connect to websites on the Tranco list in

3.3% of cases. Additionally, our manual heuristic for identifying UIDs may miss UIDs that are generated by concatenating natural language words; we expect this case to be so rare as to be almost nonexistent.

6.7 Countermeasures

6.7.1 Existing mitigations

Defending against UID smuggling is not straightforward. Given the difficulty of designing defenses that do not degrade user experience, most defenders (whether browsers or browser extensions) have so far opted for either heuristic-based or blocklist-based approaches.

Safari

Safari uses heuristics: the browser will delete cookies and website data set by a redirector unless the user also interacts with the redirector as a first-party website [114]. Safari labels an originator as performing UID smuggling if 1) it automatically redirects the user to another site, and 2) it did not receive a user activation [194, 225]. Safari also classifies a site as a UID smuggler if it participates in a navigation path that contains another known UID smuggler.

Firefox

In contrast, Firefox defends against UID smugglers using the Disconnect Tracker Protection blocklist [116, 159]. Firefox clears all storage from sites on the Disconnect tracking list after 24 hours, unless the user has loaded the site as a first party in the previous 45 days [194]. Unfortunately, we found that many UID smugglers are not yet present on the Disconnect list.

Brave

The Brave browser has multiple approaches for preventing UID smuggling. First, if the browser is navigating to a link with a query parameter for another destination URL, Brave will simply redirect to the URL in the query parameter [193]. If the browser cannot detect the final destination of the navigation, it allows the navigation to proceed, but inserts an interstitial that

warns users they will be tracked if they continue. Brave also maintains a list of UID smuggling URLs created from crowd-sourced and open-source information, as well as a blocklist of query parameter names that are commonly used for UID smuggling [195, 201]. Finally, Brave clears the storage areas associated with any sites it classifies as UID smugglers as soon as the user closes the tab that loaded them.

Chrome

While Chrome is in the process of deprecating third-party cookies [191], it does not appear to implement any features to defend against UID smuggling yet.

Extensions

Some browser extensions have begun to implement protections against UID smuggling as well. For example, Privacy Badger [94] — a browser extension by the Electronic Frontier Foundation that blocks cross-site tracking — identifies when a tracker inserts a redirector into a navigation path, and extracts the destination link from the query parameter in the redirector's URL [69]. Another extension, uBlock Origin, implements an interstitial-based approach similar to Brave's [160]. Many browser extensions, such as Adblock, Adblock Plus, and uBlock Origin, use the EasyList and EasyPrivacy filter lists [89]. We tested the URLs that CrumbCruncher found to participate in UID smuggling against the EasyList and EasyPrivacy lists, and unfortunately only 6% of the unique URLs we found would have been blocked. This result is likely because UID smuggling is such a new technique that filter lists have not yet caught up and begun blocking the URLs that participate. Additionally, EasyList and EasyPrivacy do not yet implement filters for specific query parameters. Stripping query parameters rather than blocking entire URLs is likely to result in fewer broken pages and therefore less inconvenience to users.

6.7.2 Proposed mitigations

CrumbCruncher's data can help augment the blocklists used by privacy tools and browsers to defend against UID smuggling. We provide two contributions: first, we publish our list of

token names and trackers. This list contains the query parameter names that were used to transfer UIDs across websites, as well as the list of entities that participate in UID smuggling as redirectors. Our second contribution is the code for CrumbCruncher itself, which can be run as an almost entirely automated pipeline to continuously update blocklists of navigational trackers. A major challenge of blocklist-based defenses lies in keeping those blocklists up to date: CrumbCruncher can help perform that task with much less human intervention than systems that rely on user reports of UID smuggling. The code and list of token names and trackers is available at `https://github.com/ucsdsysnet/crumbcruncher`. We also observe that while CrumbCruncher requires far less human effort than a manually created blocklist would, it still requires some manual intervention. We suggest that an approach based on machine learning for distinguishing UIDs would be a good avenue of future work, and would allow CrumbCruncher to perform its tasks in an entirely automated manner.

6.8 Related Work

The work that is most closely related to our own is Koop et al.'s study of bounce tracking [132]. Bounce tracking is similar to UID smuggling in that users' navigation paths are modified to insert redirectors that can store values as first parties, but differs in that no UIDs are transferred across contexts. Koop et al. study bounce tracking only, and do not measure whether UIDs are transferred across contexts. CrumbCruncher also clicks both iframes and anchors, whereas Koop et al.'s crawler clicks only anchors. As a result, CrumbCruncher can detect UID smuggling used by advertisements in iframes.

To verify that CrumbCruncher crawled a reasonable sample of the Web and successfully detected modified navigation paths, we measured the instances of bounce tracking that Crumb-Cruncher observed while it searched for UID smuggling, and compared our findings to the instances found by Koop et al. We found that bounce tracking that did not also involve UID smuggling was present on 2.7% of the navigation paths we studied (UID smuggling was present

on 8.1%). Because Koop et al. did not measure whether UIDs were transferred across contexts, their study labeled all UID smuggling that involved one or more redirectors as bounce tracking. Koop et al. found that "11.6% of the websites in the Alexa top 50,000 had at least one link leading to one of the top 100 redirectors" [132]. This finding seems consistent with our measurement that either UID smuggling or bounce tracking is present on a total of 10.8% (8.1% UID smuggling and 2.7% bounce tracking) of the unique navigation paths we followed.

6.8.1 Prior work on differentiating UIDs

Multiple groups have attempted to differentiate between identifiers that are capable of tracking users (UIDs) and identifiers that are not. To be a UID, a value must differ across different users, remain the same for the same user (i.e., it must not be a session ID), and contain sufficient entropy. Techniques for making these three determinations vary.

Prior work, which focused on cookies that might be UIDs, determined whether a cookie varied across users by directly or indirectly simulating different users across different crawls. Some work used two crawlers that visited the same sites simultaneously [85, 86, 93], while others simulated multiple users sequentially using a single crawler [132] or multiple crawlers [208]. Simulating multiple users sequentially enables a crawler to simulate more different users, because keeping multiple crawlers synchronized becomes more difficult as the number of crawlers increases, and a single crawler can evade this problem entirely. The disadvantage of sequential user simulation in prior work is that the crawlers did not guarantee that they visited each website more than once and thus observed each cookie more than once. Consequently, some of the cookies measured by the single sequential crawlers could not be compared across multiple users. In contrast, CrumbCruncher makes a concerted effort to visit every website in each crawl with four crawlers that represent three different users, which increases the chance that we can compare cookies and local storage values across users.

Determining whether a token is a UID also requires discarding session IDs. Most past studies labeled cookies as session IDs if their lifetime was less than a specific time, such as 90

days [132, 86, 85] or a month [29]. These works also required that the token not vary during the crawl. In contrast, Fouad et al. did not put a lifetime limit on cookies, arguing that trackers can easily link short-lived cookies on their servers [93]. We improve on prior work for discarding session IDs by immediately repeating every crawl step using a crawler that mimics one previous user. We only assume a token is a session ID if it differs across these two crawls. As detailed in Section 6.3.7, This technique allowed us to include the 16% of UID smuggling instances that we would have discarded if CrumbCruncher had used a 90-day minimum lifetime.

A further difference between CrumbCruncher and prior work is in how we determine if tokens are "the same" across users. Some previous work used the Ratcliff/Obershelp algorithm [49] to compare cookie values and allowed those values to differ by 33% [132, 85, 29], 45% [86], or by an unspecified amount [208], while still treating the cookies as "the same." We chose to discard tokens as non-UIDs only when they are entirely identical across different users, because we wished to be unambiguous about why we had discarded a particular potential UID. Some previous work also required cookie lengths to remain the same across crawls [29, 86, 208] or to only differ by 25% [132], as well as requiring cookie lengths to be at least eight characters. We require token lengths to be greater than or equal to eight characters, but we do not place any restrictions on the similarity of token lengths across users.

6.8.2 Related work on cookie syncing

A related technique to UID smuggling is cookie syncing, which has been investigated by multiple groups [208, 29, 168, 85, 169, 209]. Cookie syncing is not the same as UID smuggling, because it does not allow third parties to share a UID across top level sites when partitioned storage is in use. Instead, cookie syncing allows third parties on the same site to share a UID with each other.

6.8.3 Other related work

Trackers may circumvent partitioned storage protections using techniques that do not rely on UID smuggling, such as CNAME cloaking [76, 72] or browser fingerprinting [82, 162, 117].

CNAME cloaking is the procedure of mapping a website subdomain to a third party domain using a DNS CNAME record. This technique allows trackers to share their first party cookies, because the browser is tricked into attaching cookies from the original website's subdomain rather than the third party domain the subdomain redirects to [76]. Trackers can access session cookies, even those belonging to financial institutions, using this technique [37, 182].

Browser fingerprinting is another technique used by trackers to circumvent partitioned storage and track users across websites. Browser fingerprinting allows a tracker to use features of a user's browser such as window size, installed fonts, supported codecs, and more to create a unique "fingerprint" of that user that can function as (or generate) a UID [82]. A 2013 study crawled 20 pages for each of the Alexa top 10,000 sites and found that 40 performed browser fingerprinting [162]. A more recent study improved detection of fingerprinting code by using machine learning [117]. They then measured the Alexa top 100,000 sites and found that 10 percent of them perform fingerprinting. They find fingerprinting is more common with popular sites, as almost 25% of the Alexa top 10,000 sites perform fingerprinting.

6.9 Summary

In this chapter, we present the first systematic study of UID smuggling, a technique that allows trackers to evade browsers' protections against cross-website tracking. We find that UID smuggling is present across 8.1% of the navigations paths we observed. We publish a list of the entities that participate in UID smuggling, and classify these entities according to their behavior and purposes. Our findings can be used by browsers to improve protections against UID smuggling. Understanding the scope of UID smuggling, and the techniques by which it is conducted, is important to continue improving privacy on the Web. Browsers are increasingly

(though not yet universally) trying to protect their users from being tracked. Understanding how trackers are circumventing new browser privacy protections is important, to make sure privacy improvements are not lost as quickly as they are gained.

However, our findings indicate that trackers already frequently employ UID smuggling, even though the defense it is designed to evade (partitioned storage) is relatively recent. We conclude that partitioned storage may not be an effective intervention against web trackers, because the resources it targets (global user identifiers) are easily replaceable. If trackers can simply link per-website partitioned namespaces to approximate a global namespace, browsers will not be able to prevent trackers from building user profiles across websites.

Chapter 6, in part, is a reprint of the material as it appears in *Proceedings of the Internet Measurement Conference 2022*. Audrey Randall, Peter Snyder, Alisha Ukani, Stefan Savage, Geoffrey M. Voelker, and Aaron Schulman. The dissertation author was the primary investigator and author of this paper.

Table 6.3. The most common redirectors observed in unique domain paths. Here, "count" refers to the number of unique navigation paths the domain appeared in. *Multi-purpose smuggler.

Redirector	Count	% Domain Paths
adclick.g.doubleclick.net	36	11.2
googleads.g.doubleclick.net	20	6.2
advance.lexis.com*	10	3.1
d.agkn.com	9	2.8
btds.zog.link	9	2.8
ad.doubleclick.net	8	2.5
gm.demdex.net	8	2.5
www.kinopoisk.ru*	7	2.2
secure.jbs.elsevierhealth.com	6	1.9
t.myvisualiq.net	6	1.9
11173410.searchiqnet.com	6	1.9
optout.hearstmags.com*	6	1.9
signin.lexisnexis.com*	6	1.9
trc.taboola.com	5	1.6
l.instagram.com*	5	1.6
ads.adfox.ru*	5	1.6
www.facebook.com*	5	1.6
reseau.umontreal.ca*	5	1.6
l.facebook.com	4	1.2
rtb-use.mfadsrvr.com	4	1.2
www.campaignmonitor.com*	4	1.2
6102.xg4ken.com*	4	1.2
swallowcrockerybless.com*	4	1.2
montreal.imodules.com*	4	1.2
www.getfeedback.com*	4	1.2
kuwosm.world.tmall.com*	4	1.2
www.awin1.com	3	0.9
www.zenaps.com	3	0.9
pr.ybp.yahoo.com	3	0.9
go.dgdp.net	3	0.9
other redirectors	45	14.0

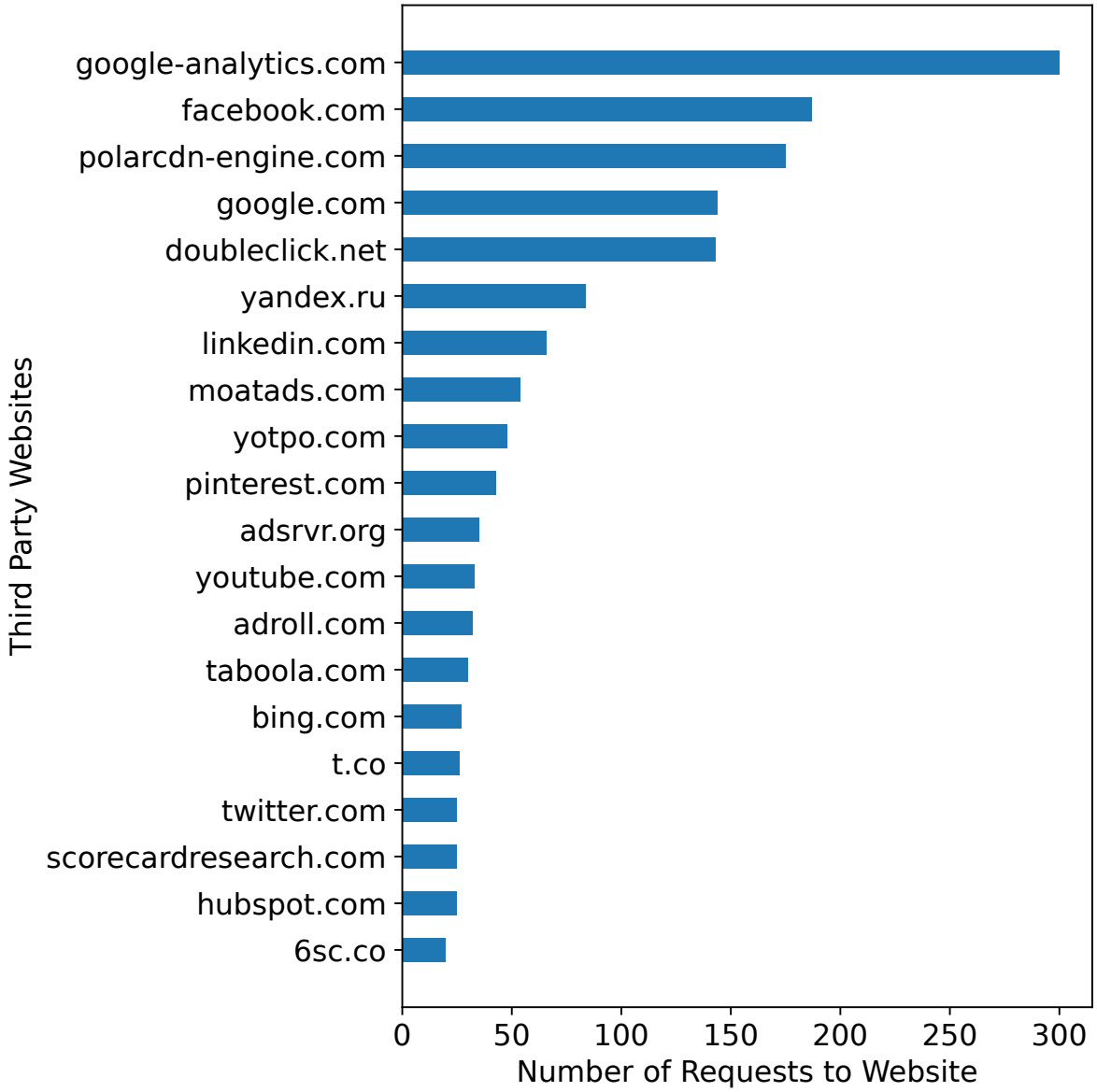

Figure 6.6. Most common domains of third party web requests sent from the destination site.

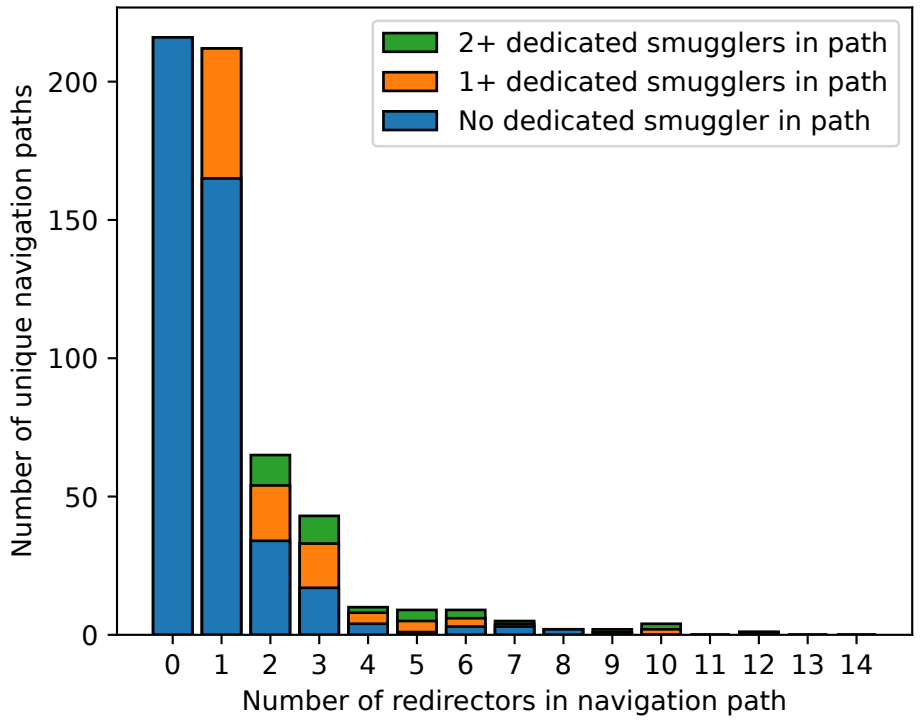

Figure 6.7. Distribution of types of redirectors in URL paths.

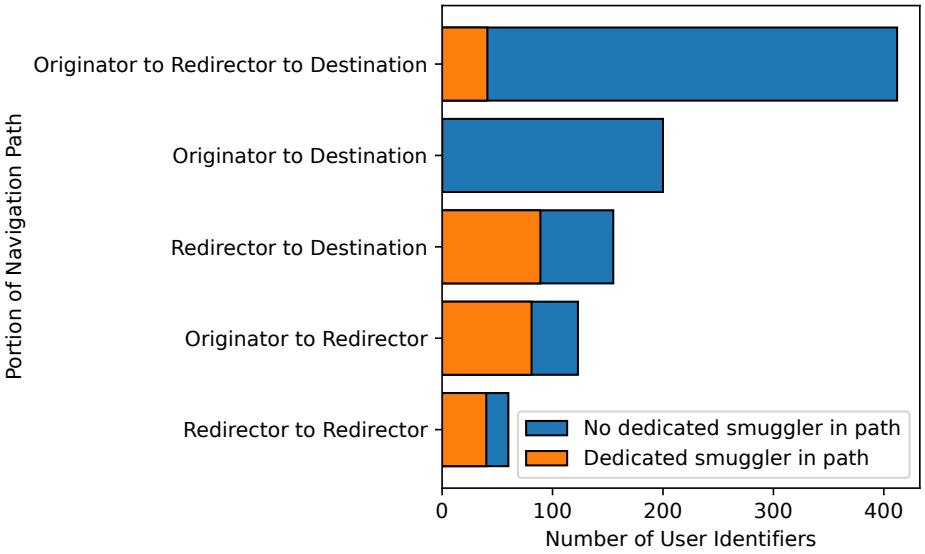

Figure 6.8. Counts of UIDs that traversed each portion of a URL path.

Chapter 7

Conclusion

This dissertation has outlined a number of methods for measuring the behavior of adversaries by studying their use of naming systems. This strategy in general — and the tools I have presented in particular — give defenders several advantages. Studying information leaked by naming systems can often provide insights even when all traditional measurement techniques are impossible. Better yet, the techniques I propose rarely require access to any proprietary resources. Both of these properties are valuable for defenders who must use limited resources to counter adversaries that hide their behavior. But even further than simply developing tools that each only work in one specific situation, I present a new strategy for measuring and combating adversaries. The steps that I took to use naming systems to measure adversary behavior and design effective interventions can be applied to new problems, new adversaries, and new naming systems in the future.

I design my measurement techniques by performing a thorough study of a naming system and looking for locations where information might leak. I and future defenders can use the same process with hitherto unexplored aspects of naming system protocols. For example, EDNS Client Subnet (ECS) is an extension of the DNS protocol that allows resolvers to give different answers based on the subnet of the requesting client's IP address. It is conceivable — perhaps even likely — that ECS might leak information about the locations of adversaries. As another example, some web services generate unique, per-user domains for various purposes, such as

"magic links" for authentication. Enumerating these unique subdomains would give a strong signal about how many people are using a service.

Similarly, my techniques can be applied to new adversaries as well as new naming systems and protocols. The list of adversaries I have not had time to study in five short years is extensive, but near the top are ransomware operators. Ransomware operators use cryptocurrencies as a payment medium, and cryptocurrencies are often vulnerable to de-anonymization attacks [152]. While ransomware operators are constantly improving their attempts to tumble their ill-gotten gains and obscure the traces of their use of blockchains, defenders are also improving their attempts to cluster wallet addresses and identify address owners. The challenging process of clustering wallet addresses may be aided by the rise of blockchain-based naming systems. As naming systems like ENS and Unstoppable Domains become more widely used to label blockchain wallets, defenders can use that signal as a clue to identify more blockchain users. The more blockchain users can be identified, the more ransomware operators and other bad actors will stand out, and the easier it will be for defenders to identify them as well.

During the course of my studies, I have also identified areas where further improvements could be made to make my work more effective and efficient. In particular, one challenge that was common to several of the studies I performed lay in distributing measurement queries from a wide variety of locations. I used two existing distributed measurement platforms for this purpose: CAIDA's Archipelago (Ark) Project [53] and RIPE's Atlas nodes [198]. Both systems were invaluable, but each has specific strengths and weaknesses. The Ark monitors allow users to run arbitrary code, but there are very few of them compared to the RIPE Atlas probes. RIPE Atlas probes, while much more numerous than Ark monitors, limit the queries that participants may make, and are primarily deployed in residential networks. A greater diversity of networks and number of participants would be useful for many measurement studies, such as the one I outline in Chapter 4.

In this dissertation, I have described how I measured the prevalence of overt stalkerware using DNS cache snooping, located DNS redirection attacks by comparing the results of CHAOS

and bogon queries, designed interventions against malware that abuses blockchains, and measured the prevalence of UID smuggling using web crawling. I have made progress both towards measuring and combating specific adversaries, and developing techniques that can be applied to future threats. As with all doctoral research, much remains to be done, but I hope that my work has aided defenders and researchers in studying and battling online attackers. I leave you, dear reader, with heartfelt thanks for making it to this page, and also a healthy skepticism that it was the best use of your time, but that is not for me to decide. Go forth, be free, enjoy some fresh air, and move on to whatever is next: just as I am now about to do, and not without regret, because I leave behind a wonderful, worthwhile, and fulfilling five years of my life.

Bibliography

[1] abuse.ch | .bit - The Next Generation of Bulletproof Hosting. https://abuse.ch/blog/dot-bit-the-next-generation-of-bulletproof-hosting/.

[2] Access Handshake names. https://learn.namebase.io/starting-from-zero/how-to-access-handshake-sites.

[3] AlienVault - Open Threat Exchange. https://otx.alienvault.com/pulse/60fd3f0d396edd67255e401f.

[4] Anomali Blog: The InterPlanetary Storm: New Malware in Wild Using InterPlanetary File System's (IPFS) p2p network. https://tinyurl.com/3tkrxnr9.

[5] Decentralized Internet for a Free Future | Skynet Labs. https://skynetlabs.com/.

[6] Ethereum Name Service. https://ens.domains.

[7] FAQ | Go Ethereum. https://geth.ethereum.org/docs/faq#wait-so-i-cant-use-fast-sync-on-an-hdd.

[8] Handshake. https://handshake.org.

[9] IPFS Powers the Distributed Web. https://ipfs.io.

[10] Namecoin. https://www.namecoin.org/.

[11] Node RPC getnames Response Exceeds Limit · Issue #447 · handshake-org/hsd. https://github.com/handshake-org/hsd/issues/447.

[12] Official website of Emercoin. https://emercoin.com/en/.

[13] OpenNIC Project. https://www.opennic.org/.

[14] OpenNIC Public Servers. https://servers.opennicproject.org/.

[15] P2P Network — Bitcoin. https://developer.bitcoin.org/devguide/p2p_network.html.

[16] Resolve Methods for Unstoppable Domains brave/brave-browser Wiki. https://github.com/brave/brave-browser.

[17] Should OpenNIC drop support for NameCoin [OpenNIC Wiki]. https://wiki.opennic.org/votings/drop_namecoin.

[18] The DGAs of Necurs. https://bin.re/blog/the-dgas-of-necurs/.

[19] The Shadowserver Foundation. https://www.shadowserver.org/.

[20] This unusual Windows malware is controlled via a P2P network. https://www.zdnet.com/article/this-unusual-windows-malware-is-controlled-via-a-p2p-network/.

[21] Malware-Traffic-Analysis.net - 2016-12-27 - EITest Rig-E from 185.156.173.99 sends Chthonic banking Trojan, December 2016. https://www.malware-traffic-analysis.net/2016/12/27/index.html.

[22] A Baza Valentine's Day | Proofpoint US, February 2021. https://www.proofpoint.com/us/blog/threat-insight/baza-valentines-day.

[23] Blockchain-DNS.info – Blockchain Name Resolver, June 2021. https://web.archive.org/web/20210621223622/https://blockchain-dns.info/#laghaus.

[24] GitHub - B-DNS/Resolver: Resolver implementation, February 2021. https://web.archive.org/web/20210228044252/https://github.com/B-DNS/Resolver/.

[25] Own research, what can open source tell us? / Sudo Null IT News, June 2021. https://web.archive.org/web/20210625041125/https://sudonull.com/post/3301-Own-research-what-can-open-source-tell-us.

[26] PRQ - Colocation, Dedicated Servers, Web hosting, VPN Tunnels, Privacy services., June 2021. https://web.archive.org/web/20210623012111/https://prq.se/.

[27] ethereum/go-ethereum, September 2022. https://github.com/ethereum/go-ethereum/blob/511bf8f18801520bf4e0c7e4d098a17d0665bc89/params/bootnodes.go.

[28] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2006.

[29] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 674–689, 2014.

[30] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. Comparing DNS Resolvers in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2010.

[31] Ron Aitchison. *Pro Dns and BIND 10*. Apress, 2011.

[32] Hüseyin Akcan, Torsten Suel, and Hervé Brönnimann. Geographic Web Usage Estimation By Monitoring DNS Caches. In *Proceedings of the International Workshop on Location and the Web (LOCWEB)*, 2008.

[33] Rami Al-Dalky, Michael Rabinovich, and Kyle Schomp. A Look at the ECS Behavior of DNS Resolvers. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019.

[34] Rami Al-Dalky and Kyle Schomp. Characterization of Collaborative Resolution in Recursive DNS Resolvers. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2018.

[35] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael B. Abu-Ghazaleh. Collaborative Client-Side DNS Cache Poisoning Attack. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[36] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. Blockstack: A global naming and storage system secured by blockchains. In *2016 USENIX annual technical conference (USENIX ATC 16)*, pages 181–194, 2016.

[37] Assel Aliyeva and Manuel Egele. Oversharing Is Not Caring: How CNAME Cloaking Can Expose Your Session Cookies. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, pages 123–134, 2021.

[38] S. Alrwais, X. Liao, X. Mi, P. Wang, X. Wang, F. Qian, R. Beyah, and D. McCoy. Under the Shadow of Sunshine: Understanding and Detecting Bulletproof Hosting on Legitimate Service Provider Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 805–823, 2017.

[39] Narseo Vallina-Rodriguez andSrikanth Sundaresan andChristian Kreibich andNicholas Weaver andVern Paxson. Beyond the Radio: Illuminating the Higher Layers of Mobile Networks. In *Proceedings of the ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 375–387, Florence, Italy, 2015.

[40] Anonymous. The Collateral Damage of Internet Censorship by DNS Injection. In *Proceedings of the ACM SIGCOMM*, pages 21–27, Helsinki, Finland, June 2012.

[41] Anonymous. Towards a Comprehensive Picture of the Great Firewall's DNS Censorship. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, San Diego, CA, USA, 2014.

[42] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. *USENIX Security*, page 16, 2012.

[43] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet Censorship in Iran: A First Look. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, Washington, D.C., USA, 2013.

[44] aslushnikov. Intercept target creation: Issue #3667, December 2018. https://github.com/puppeteer/puppeteer/issues/3667.

[45] Johannes Bader. Yet Another Bazar Loader DGA, January 2021. https://bin.re/blog/yet-another-bazarloader-dga/.

[46] Jeff Baumgartner. Comcast Taps Arris, Technicolor for 'XB6' Gateways: Sources. https://www.nexttv.com/news/comcast-taps-arris-technicolor-xb6-gateways-sources-409944, May 2021.

[47] berstend. Target creation event listeners are sometimes not executed early enough: Issue #2669, June 2018. https://github.com/puppeteer/puppeteer/issues/2669.

[48] Chetna Bindra. Building a privacy-first future for web advertising, January 2021. https://blog.google/products/ads-commerce/2021-01-privacy-sandbox/.

[49] Paul E. Black. Ratcliff/Obershelp pattern recognition, January 2021. https://www.nist.gov/dads/HTML/ratcliffObershelp.html.

[50] Marc Blanchet and Lars-Johan Liman. RFC 7720: DNS Root Name Service Protocol and Deployment Requirements, 2015.

[51] Danny Bradbury. Testing the defences of bulletproof hosting companies. *Network Security*, 2014(6):8–12, 2014.

[52] Andrew Brandt. BazarLoader deploys a pair of novel spam vectors, April 2021. https://news.sophos.com/en-us/2021/04/15/bazarloader/.

[53] CAIDA. Archipelago (ark) Measurement Infrastructure, 06 2020. https://www.caida.org/projects/ark/.

[54] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the Performance of an Anycast CDN. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2015.

[55] Patricia Callejo, Rubén Cuevas, Narseo Vallina-Rodriguez, and Ángel Cuevas. Measuring the Global Recursive DNS Infrastructure: A View From the Edge. In *Proceedings of the IEEE Access*, 2019.

[56] Fran Casino, Nikolaos Lykousas, Vasilios Katos, and Constantinos Patsakis. Unearthing malicious campaigns and actors from the blockchain DNS ecosystem. *Computer Communications*, pages 217–230.

[57] Sebastian Castro, Duane Wessels, Marina Fomenkov, and Kimberly Claffy. A Day at the Root of the Internet. *ACM Computer Communication Review (CCR)*, pages 41–46, 2008.

[58] RDK Central. CcspXDNS, 04 2021. https://wiki.rdkcentral.com/display/RDK/CcspXDNS.

[59] RDK Central. RDK Surpasses 80 Million Device Deployments Across Leading Video and Broadband Service Providers. https://rdkcentral.com/rdk-surpasses-80-million-device-deployments-across- leading-video-and-broadband-service-providers/, May 2021.

[60] RDK Central. Source code for RDK-B, 05 2021. https://code.rdkcentral.com/r/plugins/gitiles/rdkb/components/opensource/ccsp/Utopia/+/7afe5aba7c8e9b89a182cdcffe16159c3b431b16/source/firewall/firewall.c.

[61] Rahul Chatterjee, Periwinkle Doerfler, Hadas Orgad, Sam Havron, Jackeline Palmer, Diana Freed, Karen Levy, Nicola Dell, Damien McCoy, and Thomas Ristenpart. The Spyware Used in Intimate Partner Violence. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 441–458, 2018.

[62] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of the ACM SIGCOMM*, 2015.

[63] Qi Alfred Chen, Eric Osterweil, Matthew Thomas, and Z. Morley Mao. MitM Attack by Name Collision: Cause Analysis and Vulnerability Assessment in the New gTLD Era. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 675–690, May 2016. ISSN: 2375-1207.

[64] Chuck. "listen on 5353 too?", 2011. https://groups.google.com/g/public-dns-discuss/c/MfpyYHcqzjI.

[65] Taejoong Chung, David Choffnes, and Alan Mislove. Tunneling for Transparency: A Large-Scale Analysis of End-to-End Violations in the Internet. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 199–213, Santa Monica California USA, November 2016.

[66] Robert Clarke and Thomas Lancaster. Eliminating the successor to plagiarism? Identifying the usage of contract cheating sites. In *Proceedings of the International Plagiarism Conference*, 2006.

[67] Hacker Codex. How to Stop Your ISP from Hijacking Your DNS Servers, 2012. https://hackercodex.com/guide/how-to-stop-isp-dns-server-hijacking/.

[68] Tom Creighton, Chris Griffiths, Jason Livingood, and Ralf Weber. DNS Redirect Use by Service Providers. Internet Draft: draft-livingood-dns-redirect-03, October 2010.

[69] Bennett Cyphers. Privacy Badger Rolls Out New Ways to Fight Facebook Tracking, May 2018. https://www.eff.org/deeplinks/2018/05/privacy-badger-rolls-out-new-ways-fight-facebook-tracking.

[70] Bennett Cyphers. Google's FLoC Is a Terrible Idea, March 2021. https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea.

[71] David Dagon, Niels Provos, Christopher P. Lee, and Wenke Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2008.

[72] Ha Dao, Johan Mazel, and Kensuke Fukuda. Characterizing CNAME cloaking-based tracking on the web. In *Proceedings of the IFIP/IEEE Traffic Measurement Analysis Conference (TMA)*, 2020.

[73] Wouter B. de Vries, Roland van Rijswijk-Deij, Pieter-Tjerk de Boer, and Aiko Pras. Passive Observations of a Large DNS Service: 2.5 Years in the Life of Google. pages 190–200, 2019.

[74] Selena Deckelmann. Firefox continues push to bring DNS over HTTPS by default for US users. 2020. https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/.

[75] Frank Denis. Performance: How Long Does a Second Actually Last?, 2012. https://dzone.com/articles/performance-how-long-does.

[76] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. The CNAME of the Game: Large-scale Analysis of DNS-based Tracking Evasion. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, pages 394–412, 2021.

[77] Dnsmasq. Dnsmasq - network services for small networks., 05 2021. https://thekelleys.org.uk/dnsmasq/doc.html.

[78] DomainTools. Iris Investigation Platform - Passive DNS, January 2022. https://www.domaintools.com/products/iris.

[79] Yuhao Dong, Woojung Kim, and Raouf Boutaba. Bitforest: a portable and efficient blockchain-based naming system. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 226–232. IEEE, 2018.

[80] Yuhao Dong, Woojung Kim, and Raouf Boutaba. Conifer: centrally-managed PKI with blockchain-rooted trust. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1092–1099. IEEE, 2018.

[81] Haixin Duan, Nicholas Weaver, Zongxu Zhao, Meng Hu, Jinjin Liang, Jian Jiang, Kang Li, and Vern Paxson. Hold-On: Protecting Against On-Path DNS Poisoning. In *Proceedings of the Workshop on Securing and Trusting Internet Names (SATIN)*, London, United Kingdom, 2012.

[82] Peter Eckersley. How Unique is Your Web Browser? In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, page 1–18, 2010.

[83] The Economist. France v Google, 2013. https://www.economist.com/business/2013/01/12/france-v-google.

[84] RANDI EITZMAN, KIMBERLY GOODY, and JESSA VALDEZ. How the Rise of Cryptocurrencies Is Shaping the Cyber Crime Landscape: Blockchain Infrastructure Use. https://www.mandiant.com/resources/blog/how-rise-cryptocurrencies-shaping-cyber-crime-landscape-blockchain-infrastructure-use.

[85] Steven Englehardt and Arvind Narayanan. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1388–1401, 2016.

[86] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies That Give You Away: The Surveillance Implications of Web Tracking. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 289–299, 2015.

[87] etherscan.io. Ethereum Transaction Hash (Txhash) Details | Etherscan. http://etherscan.io/tx/0x66d1300ef612ec633ede2f26fca4a17d5321ee4c0642d111b5092ec2efae90ba.

[88] Xun Fan, John Heidemann, and Ramesh Govindan. Evaluating Anycast in the Domain Name System. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2013.

[89] Fanboy, MonztA, Famlam, and Khrin. EasyList - Overview, September 2022. https://easylist.to/.

[90] Oliver Farnan, Alexander Darer, and Joss Wright. Analysing Censorship Circumvention with VPNs Via DNS Cache Snooping. In *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, 2019.

[91] Rom Feria. Hiding from Data Collectors. https://rom.feria.name/hiding-from-data-collectors-9485dcb93b22.

[92] Xfinity Community Forum. Changing the DNS on my machine didn't work..., 2019. https://forums.xfinity.com/t5/Your-Home-Network/Changing-the-DNS-on-my-machine-didn-t-work/m-p/3268054#M309013.

[93] Imane Fouad, Nataliia Bielova, Arnaud Legout, and Natasa Sarafijanovic-Djukic. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, pages 499–518, 2020.

[94] Electronic Frontier Foundation. Privacy Badger, February 2022. https://privacybadger.org/.

[95] Diana Freed, Sam Havron, Emily Tseng, Andrea Gallardo, Rahul Chatterjee, Thomas Ristenpart, and Nicola Dell. "is my phone hacked?" Analyzing Clinical Computer Security Interventions with Survivors of Intimate Partner Violence. In *Proceedings of the ACM Conference on Human-Computer Interaction*, 2019.

[96] Vinay Goel. Get to know the new Topics API for Privacy Sandbox, January 2022. https://blog.google/products/chrome/get-know-new-topics-api-privacy-sandbox/.

[97] Google. Google Public DNS: Performance Benefits, 2018. https://developers.google.com/speed/public-dns/docs/performance?hl=zh-cn.

[98] Google. Google Public DNS FAQ. 2020. https://developers.google.com/speed/public-dns/faq#isp.

[99] Luis Grangeia. DNS Cache Snooping or Snooping the Cache for Fun and Profit. Technical report, Securi Team-Beyond Security, 2004.

[100] Yunhong Gu. Google Public DNS and Location-Sensitive DNS Responses. https://webmasters.googleblog.com/2014/12/google-public-dns-and-location.html, 2014.

[101] Ólafur Guðmundsson. Introducing DNS Resolver, 1.1.1.1 (not a joke). https://blog.cloudflare.com/dns-resolver-1-1-1-1/.

[102] Peter Hamilton. Server-to-Server Tracking Basics (Web-Based Affiliate Marketing), April 2012. https://www.tune.com/blog/server-side-tracking-basics/.

[103] Fakhar ul Hassan, Anwaar Ali, Mohamed Rahouti, Siddique Latif, Salil Kanhere, Jatinder Singh, AlaAl-Fuqaha, Umar Janjua, Adnan Noor Mian, Junaid Qadir, and Jon Crowcroft. Blockchain And The Future of the Internet: A Comprehensive Review, November 2020. http://arxiv.org/abs/1904.00733.

[104] Seppo Hätönen, Aki Nyrhinen, Lars Eggert, Stephen Strowes, Pasi Sarolahti, and Markku Kojo. An Experimental Study of Home Gateway Characteristics. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2010.

[105] Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas Ristenpart. Clinical Computer Security for Victims of Intimate Partner Violence. In *Proceedings of the USENIX Security*, 2019.

[106] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous, or: one-domain-to-rule-them-all.org. In *IEEE Conference on Communications and Network Security (CNS)*, 2013.

[107] Paul Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 9494, October 2018. https://tools.ietf.org/html/rfc8484.

[108] Michael Horowitz. OpenDNS provides added safety for free, 12 2007. https://www.cnet.com/news/opendns-provides-added-safety-for-free/.

[109] Wei-hong Hu, Meng Ao, Lin Shi, Jia-gui Xie, and Yang Liu. Review of blockchain-based DNS alternatives. *Chinese Journal of Network and Information Security*, 3(3):71–77.

[110] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016. https://tools.ietf.org/html/rfc7858.

[111] Tim Huang, Johann Hofmann, and Arthur Edelstein. Firefox 86 Introduces Total Cookie Protection, February 2022. https://blog.mozilla.org/security/2021/02/23/total-cookie-protection.

[112] Zhangrong Huang, Ji Huang, and Tianning Zang. Leopard: Understanding the threat of blockchain domain name based malware. In *International Conference on Passive and Active Network Measurement*, pages 55–70. Springer, 2020.

[113] IAB. IAB Tech Lab Content Taxonomy, February 2022. https://www.iab.com/guidelines/iab-tech-lab-content-taxonomy/.

[114] Apple Inc. Prevent cross-site tracking in Safari on Mac, 2022. https://support.apple.com/guide/safari/prevent-cross-site-tracking-sfri40732/mac.

[115] Disconnect Inc. Entity List, 2022. https://github.com/mozilla-services/shavar-prod-lists/blob/master/disconnect-entitylist.json.

[116] Disconnect Inc. Tracker Protection Lists, 2022. https://github.com/disconnectme/disconnect-tracking-protection.

[117] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1143–1161, 2021.

[118] joenathanone. Hacker News forum: Quad9 location request, 11 2017. https://news.ycombinator.com/item?id=15712940.

[119] Brian Johnson, Ivan Efremov, and Peter Snyder. Ephemeral Third-party Site Storage, February 2021. https://brave.com/privacy-updates/7-ephemeral-storage/.

[120] Ben Jones, Nick Feamster, Vern Paxson, Nicholas Weaver, and Mark Allman. Detecting DNS Root Manipulation. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2016.

[121] Jaeyeon Jung, Arthur W. Berger, and Hari Balakrishnan. Modelling TTL-based Internet Caches. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[122] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS Performance and the Effectiveness of Caching. In *Proceedings of the IEEE/ACM Transactions on Networking*, 2002.

[123] Harry Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of Namecoin and lessons for decentralized namespace design.

[124] Richard Kastelein. Blockchain Startup Nebulis Set to Solve Problem of DDoS Attacks On DNS Servers, December 2016. https://www.the-blockchain.com/2016/12/06/blockchain-startup-nebulis-set-prevent-ddos-attacks-dns-servers/.

[125] Aniket Kesari, Chris Hoofnagle, and Damon McCoy. Deterring Cybercrime: Focus on Intermediaries. *Berkeley Technology Law Journal*, 32(3):1093–1134, 2017. https://heinonline.org/HOL/P?h=hein.journals/berktech32&i=1137.

[126] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going Wild: Large-Scale Classification of Open DNS Resolvers. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2015.

[127] Seunghoe Kim. Deep analysis of KPOT Stealer, July 2021. https://medium.com/s2wblog/deep-analysis-of-kpot-stealer-fb1d2be9c5dd.

[128] Amit Klein and Benny Pinkas. DNS Cache-Based User Tracking. In *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[129] Amit Klein, Haya Shulman, and Michael Waidner. Counting in the Dark: DNS Caches Discovery and Enumeration in the Internet. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017.

[130] Jason Knight. Domain Takedowns A Step by Step Analysis for Law Enforcement. February 2015.

[131] Maria Konte, Roberto Perdisci, and Nick Feamster. Aswatch: An as reputation system to expose bulletproof hosting ases. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 625–638, 2015.

[132] Martin Koop, Erik Tews, and Stefan Katzenbeisser. In-Depth Evaluation of Redirect Tracking and Link Usage. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, pages 394–413, 2020.

[133] Ignat Korchagin and Lennart Poettering. Git commit: resolved: use Cloudflare public DNS server as a default fallback, 02 2019. https://github.com/systemd/systemd/commit/def3c7c791e7918a889c2b93dee039ab77b3a523.

[134] Mario Korf and Barb Strom. Introducing a New whoami Tool for DNS Resolver Information, 2018. https://developer.akamai.com/blog/2018/05/10/introducing-new-whoami-tool-dns-resolver-information.

[135] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the Edge Network. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2010.

[136] Thomas Lancaster and Robert Clarke. *Contract Cheating: The Outsourcing of Assessed Student Work*. 2015.

[137] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. A research-oriented top sites ranking hardened against manipulation - Tranco, 2021. https://tranco-list.eu/.

[138] Zach Lerner. MICROSOFT THE BOTNET HUNTER: THE ROLE OF PUBLIC-PRIVATE PARTNERSHIPS IN MITIGATING BOTNETS. *Harvard Journal of Law and Technology*, 28(1):26, 2014.

[139] Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Mark Felegyhazi, Chris Grier, Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. Click Trajectories: End-to-End Analysis of the Spam Value Chain. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 431–446, 2011.

[140] Abner Li. Google's Public DNS turns '8.8.8.8 years old,' teases 'exciting' future announcements. https://9to5google.com/2018/08/13/google-public-dns-8-8-8-8-years-future-announcements/, 2018.

[141] He (Lonnie) Liu, Kirill Levchenko, Márk Félegyházi, Christian Kreibich, Gregor Maier, Geoffrey M Voelker, and Stefan Savage. On the Effects of Registrar-level Intervention. page 8.

[142] Liu, Baojun and Lu, Chaoyi and Duan, Haixin and Liu, Ying and Li, Zhou and Hao, Shuang and Yang, Min. Who is Answering My Queries: Understanding and Characterizing Interception of the DNS Resolution Path. In *Proceedings of the USENIX Security*, page 1113–1128, Baltimore, MD, USA, 2018.

[143] Joe Security LLC. Automated Malware Analysis Report for bz.arm-20211111-0250 - Generated by Joe Sandbox, November 2021. https://www.joesandbox.com/analysis/519713/0/pdf.

[144] Sports Reference LLC. Sports Reference | Sports Stats, fast, easy, and up-to-date, September 2022. https://www.sports-reference.com/.

[145] Owen Lystrup. OpenDNS Enforces Threat Intelligence at the Speed of Automatic, 04 2020. https://umbrella.cisco.com/blog/opendns-custom-api-operationalizes-threat-intelligence.

[146] Internet Archive Wayback Machine. Mobile Spy App for Personal Catch Cheating Spouses, February 2018. https://web.archive.org/web/20180216084527/http://hellospy.com/hellospy-for-personal-catch-cheating-spouses.aspx?lang=en-US.

[147] Internet Archive Wayback Machine. Catch Cheating Spouses With TheTruthSpy, May 2020. https://web.archive.org/web/20200523174940/https://thetruthspy.com/catch-cheating-spouses-with-thetruthspy/.

[148] Matthew Mackie. CryptoDNS–Should We Worry? https://insights.sei.cmu.edu/blog/cryptodns-should-we-worry/.

[149] Linux Programmer's Manual. GetHostByName – Linux manual page, 06 2020. https://www.man7.org/linux/man-pages/man3/gethostbyname_r.3.html.

[150] Zhuoqing Morley Mao, Charles D. Cranor, Fred Douglis, Michael Rabinovich, Oliver Spatscheck, and Jia Wang. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *Proceedings of the USENIX Annual Technical Conference*, 2002.

[151] Andrew McGregor, Phillipa Gill, and Nicholas Weaver. Cache Me Outside: A New Look at DNS Cache Probing. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, pages 427–443, Virtual, 2021.

[152] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013.

[153] Xavier Mertens. Systemd Could Fallback to Google DNS?, 06 2017. https://isc.sans.edu/forums/diary/Systemd+Could+Fallback+to+Google+DNS/22516/.

[154] Ariana Mirian, Joe DeBlasio, Stefan Savage, Geoffrey M. Voelker, and Kurt Thomas. Hack for Hire: Exploring the Emerging Market for Account Hijacking. In *Proceedings of the International World Wide Web Conference (WWW)*, 2019.

[155] Paul V. Mockapetris. Domain Names - Implementation and Specification, 2020. https://tools.ietf.org/html/rfc1035.

[156] Joanna Moubarak, Maroun Chamoun, and Eric Filiol. Developing a K-ary malware using blockchain. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–4, April 2018. ISSN: 2374-9709.

[157] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019.

[158] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. When the Dike Breaks: Dissecting DNS Defenses During DDoS. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2018.

[159] Mozilla. Enhanced Tracking Protection in Firefox for desktop, 2022. https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop.

[160] Jared Newman. The incredibly sneaky way websites sidestep privacy tools to spy on you, August 2021. https://www.fastcompany.com/90663878/bounce-tracking-privacy-browsers-brave-firefox-safari-edge.

[161] Yu Ng. In the World of DNS, Cache is King, 07 2014. https://blog.catchpoint.com/2014/07/15/world-dns-cache-king/.

[162] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 541–555, 2013.

[163] NLNet Labs. Unbound configuration file, 2020. https://nlnetlabs.nl/documentation/unbound/unbound.conf/.

[164] Arman Noroozian, Jan Koenders, Eelco van Veldhuizen, Carlos H. Ganan, Sumayah Alrwais, Damon McCoy, and Michel van Eeten. Platforms in Everything: Analyzing Ground-Truth Data on the Anatomy and Economics of Bullet-Proof Hosting. pages 1341–1356, 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/noroozian.

[165] OpenDNS. FAQ: Why did Cisco buy OpenDNS?, 2015. https://www.opendns.com/cisco-opendns/.

[166] John S. Otto, Mario A. Sánchez, John P. Rula, and Fabián E. Bustamante. Content Delivery and the Natural Evolution of DNS: Remote DNS Trends, Performance Issues and Alternative Solutions. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2012.

[167] Nikolaos Pantazopoulos and Stefano Antenucci. In-depth analysis of the new Team9 malware family, June 2020. https://blog.fox-it.com/2020/06/02/in-depth-analysis-of-the-new-team9-malware-family/.

[168] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos Markatos. Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask. In *Proceedings of the World Wide Web Conference (WWW)*, pages 1432–1442, 2019.

[169] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. Exclusive: How the (synced) Cookie Monster breached my encrypted VPN session. In *Proceedings of the European Workshop on Systems Security (EuroSec)*, pages 1–6, April 2018.

[170] Constantinos Patsakis, Fran Casino, Nikolaos Lykousas, and Vasilios Katos. Unravelling Ariadne's Thread: Exploring the Threats of Decentralised DNS. *IEEE Access*, 8:118559–118571, 2020. Conference Name: IEEE Access.

[171] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global Measurement of DNS Manipulation. In *Proceedings of the USENIX Security*, Vancouver, BC, Canada, 2017.

[172] Rob Pegoraro. The blockchain is making domain names more private—for good or bad, October 2021. https://www.fastcompany.com/90686579/blockchain-domains-bit-microsoft.

[173] Dave Piscitello. ConfickerSummaryandReview20100507. page 18.

[174] Andreas Pitsillidis, Chris Kanich, Geoffrey M. Voelker, Kirill Levchenko, and Stefan Savage. Taster's Choice: A Comparative Analysis of Spam Feeds. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2012.

[175] Stijn Pletinckx, Cyril Trap, and Christian Doerr. Malware Coordination using the Blockchain: An Analysis of the Cerber Ransomware. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, May 2018.

[176] Stijn Pletinckx, Cyril Trap, and Christian Doerr. Malware Coordination using the Blockchain: An Analysis of the Cerber Ransomware. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2018.

[177] Quad9. Quad9 Enabled Across New York City Guest and Public WiFi, 03 2018. https://www.quad9.net/quad9-enabled-across-new-york-city-guest-and-public-wifi/.

[178] Quad9. Quad9: Internet Security And Privacy In a Few Easy Steps, 06 2020. https://www.quad9.net.

[179] Moheeb Abu Rajab, Fabian Monrose, Andreas Terzis, and Niels Provos. Peeking Through the Cloud: DNS-Based Estimation and Its Applications. In *Proceedings of the Applied Cryptography and Network Security Conference (ACNS)*, 2008.

[180] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My Botnet Is Bigger Than Yours (maybe, Better Than Yours): Why Size Estimates Remain Challenging. In *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[181] Shoushou Ren, Bingyang Liu, Fei Yang, Xinpeng Wei, Xue Yang, and Chuang Wang. BlockDNS: Enhancing Domain Name Ownership and Data Authenticity with Blockchain. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[182] Tongwei Ren, Alexander Wittman, Lorenzo De Carli, and Drew Davidson. An Analysis of First-Party Cookie Exfiltration due to CNAME Redirections. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*, 2021.

[183] Chromium Git repository. User Data Directory, 2022. https://chromium.googlesource.com/chromium/src.git/+/HEAD/docs/user_data_dir.md.

[184] Tarcan Turgut Rohprimardho and Roland M. van Rijswijk-Deij. Peeling the Google DNS Onion. Technical report, 2015.

[185] root-servers.org. Root Server Technical Operations Association homepage, 2020. https://root-servers.org/.

[186] Chris Scharff. Have problems with 1.1.1.1? *read Me First*, 2018. https://community.cloudflare.com/t/have-problems-with-1-1-1-1-read-me-first/15902.

[187] Sam Schechner, Patience Haggin, and Tripp Mickle. Google Overhauls Cookie Replacement Plan After Privacy Critiques - WSJ, January 2022. https://www.wsj.com/articles/google-overhauls-cookie-replacement-plan-after-privacy-critiques-11643115603.

[188] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D Strowes, and Narseo Vallina-Rodriguez. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 478–493, 2018.

[189] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On Measuring the Client-Side DNS Infrastructure. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2013.

[190] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. Assessing DNS Vulnerability to Record Injection. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2014.

[191] Justin Schuh. Building a more private web: A path towards making third partycookies obsolete, January 2020. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html.

[192] Lior Shafir, Yehuda Afek, Anat Bremler-Barr, Neta Peleg, and Matan Sabag. DNS Negative Caching in the Wild. In *Proceedings of the ACM SIGCOMM Conference Posters and Demos*, 2019.

[193] Peter Snyder. Debouncing, October 2021. https://brave.com/privacy-updates/11-debouncing/.

[194] Peter Snyder and Jeffrey Yasskin. Navigational-Tracking Mitigations, May 2022. https://privacycg.github.io/nav-tracking-mitigations/.

[195] Brave Software. adblock-lists/brave-lists/debounce.json, September 2022. https://github.com/brave/adblock-lists/blob/1453e599881854f970ab9164a104104ea9ec139f/brave-lists/debounce.json.

[196] Redhat Customer Solutions. systemd-resolved falls back to Google public DNS servers, 06 2017. https://access.redhat.com/solutions/3083631.

[197] Sooel Son and Vitaly Shmatikov. The Hitchhiker's Guide to DNS Cache Poisoning. In *Proceedings of the International Conference on Security and Privacy in Communication Systems (SECURECOMM)*, 2010.

[198] RIPE NCC Staff. Ripe Atlas: A Global Internet Measurement Network. *Internet Protocol Journal*, 2015.

[199] Statista. Digital advertising spending worldwide from 2021 to 2026, 2022. https://www.statista.com/statistics/237974/online-advertising-spending-worldwide/.

[200] Xfinity Help & Support. Overview of Xfinity Gateways, 2020. https://www.xfinity.com/support/articles/broadband-gateways-userguides.

[201] Brave Privacy Team. "Unlinkable Bouncing" for More Protection Against Bounce Tracking, March 2022. https://brave.com/privacy-updates/16-unlinkable-bouncing/.

[202] Microsoft Defender Security Research Team. Hunting down Dofoil with Windows Defender ATP, April 2018. https://www.microsoft.com/security/blog/2018/04/04/hunting-down-dofoil-with-windows-defender-atp/.

[203] Top Draw Team. Online Advertising Costs In 2021 | Top Draw, March 2021. https://www.topdraw.com/insights/is-online-advertising-expensive/.

[204] Threatcrowd.org. Malware: Win32.Trojan-dropper.Necurs. https://www.threatcrowd.org/listMalware.php?antivirus=Win32.Trojan-dropper.Necurs.Dzts.

[205] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a Haystack: Tracking Down Elite Phishing Domains in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, October 2018.

[206] Ravikant Tiwari. Evolution of GandCrab Ransomware, August 2018. https://www.acronis.com/en-us/blog/posts/gandcrab/.

[207] Trellix. Trellix Insights: GandCrab ransomware version 2 released with new .crab extension and other changes, August 2022. https://kcm.trellix.com/corporate/index?page=content&id=KB93103&locale=en_US.

[208] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. The Unwanted Sharing Economy: An Analysis of Cookie Syncing and User Transparency under GDPR. In *arXiv preprint arXiv:1811.08660*, 2018.

[209] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. Measuring the Impact of the GDPR on Data Sharing in Ad Networks. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, pages 222–235, October 2020.

[210] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*, San Diego, CA, 2020.

[211] Paul Vixie. What DNS is not. *Communications of the ACM*, 52(12):43–47, 2009.

[212] SoftEther VPN. VPNGate: Public VPN Relay Servers, 2021. https://vpngate.net.

[213] Jane Wakefield. Google slammed over ad-cookie replacement flip-flop. *BBC News*, January 2022. https://www.bbc.com/news/technology-60138876.

[214] Mary Walker and Cynthia Townley. Contract cheating: a new challenge for academic honesty? *Journal of Academic Ethics*, 10(1):27–44, March 2012.

[215] David Y. Wang, Matthew Der, Mohammad Karami, Lawrence Saul, Damon McCoy, Stefan Savage, and Geoffrey M. Voelker. Search + Seizure: The Effectiveness of Interventions on SEO Campaigns. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 359–372, Vancouver, BC, Canada, 2014. ACM Press. http://dl.acm.org/citation.cfm?doid=2663716.2663738.

[216] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. In *Proceedings of the USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[217] Nicholas Weaver, Christian Kreibich, Boris Nechaev, and Vern Paxson. Implications of Netalyzr's DNS Measurements. In *Proceedings of the Workshop on Securing and Trusting Internet Names (SATIN)*, 2011.

[218] Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting DNS for Ads and Profit. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, San Francisco, CA, USA, 2011.

[219] WebKit. Tracking Prevention Policy, August 2019. https://webkit.org/tracking-prevention-policy/.

[220] Webshrinker. IAB Categories, 2022. https://docs.webshrinker.com/v3/iab-website-categories.html#iab-categories.

[221] Webshrinker. Webshrinker Website, 2022. https://www.webshrinker.com/.

[222] Lan Wei and John S. Heidemann. Whac-A-Mole: Six Years of DNS Spoofing. *arXiv*, 2020. https://arxiv.org/abs/2011.12978.

[223] David P. Wiggins. Xvfb—virtual framebuffer X server for X Version 11, May 2022. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

[224] John Wilander. Intelligent Tracking Prevention 2.3, September 2019. https://webkit.org/blog/9521/intelligent-tracking-prevention-2-3/.

[225] John Wilander. Bounce Tracking Protection · Issue #6 · privacycg/proposals, February 2020. https://github.com/privacycg/proposals/issues/6.

[226] Craig E. Wills, Mikhail Mikhailov, and Hao Shang. Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2003.

[227] Suzanne Woolf and David Conrad. Requirements for a Mechanism Identifying a Name Server Instance. RFC 4892, June 2007. https://tools.ietf.org/html/rfc4892.

[228] Pengcheng Xia, Haoyu Wang, Zhou Yu, Xinyu Liu, Xiapu Luo, Guoai Xu, and Gareth Tyson. Challenges in Decentralized Name Management: The Case of ENS. In *Proceedings of the 2013 Internet Measurement Conference (IMC)*, 2022.