DNS Prefetching and Its Privacy Implications:
When Good Things Go Bad

Srinivas Krishnan and Fabian Monrose
Department of Computer Science
University of North Carolina at Chapel Hill,
{*krishnan,fabian*}*@cs.unc.edu*

Abstract

A recent trend in optimizing Internet browsing speed is to optimistically pre-resolve (or prefetch) DNS resolutions. While the practical benefits of doing so are still being debated, this paper attempts to raise awareness that current practices could lead to privacy threats that are ripe for abuse. More specifically, although the adoption of several browser optimizations have already raised security concerns, we examine how prefetching amplifies disclosure attacks to a degree where it is possible to infer the likely *search terms* issued by clients using a given DNS resolver. The success of these inference attacks relies on the fact that prefetching inserts a significant amount of context into a resolver's cache, allowing an adversary to glean far more detailed insights than when this feature is turned off.

1 Introduction

Access to information at our finger tips is a luxury we have come to expect. We all have become impatient, continually demanding faster answers to our questions—be it for the best remedies to our current ailment, directions to a weekend getaway, the best prices for that must have item, recommendations for that restaurant we just drove by, etc. All too often, we turn to our favorite search engine, hopeful that it can immediately quench our thirst for knowledge. In turn, software engineers and architects are continuously challenged with finding ways to improve responsiveness on the Web, and help us quickly wade through the deluge of responses. Of late, a growing trend in optimizing the speed browsers is to *pre-resolve* (or prefetch) the DNS resolution of domains in hyperlinks so that they are ready to be served in the off chance that the user decides to click on them.

While the idea of pre-resolving domain names is by no means new, it is somewhat surprising that it has only recently caught on [1]. The concept appears to have been first proposed (at least in the academic literature) by Cohen and Kaplan [2] as a low-overhead alternative to prefetching of documents. The key observation is that since DNS resolutions are dominated by latency, one way to decrease user-perceived delay is to preform speculative pre-resolution. The improvement in performance comes from the fact that resolving a DNS query often involves communication with at least one remote nameserver, and in some cases, may require following referral chains across several servers—a task that could take several seconds to complete.

Loosely speaking, the strategies being applied by the browsers we examined involve pre-resolving all hyperlinks on a page while its being loaded, and optimistically pre-resolving names as a user types in the navigation or search bar. For the most part, the goal is to strike a balance between the number of eliminated cache misses and the overhead of generating additional queries. One common, and prudent, restriction appears to be the disabling of prefetching of hyperlinks appearing in `HTTPS` pages; apparently to prevent an eavesdropper from learning information in a context where confidentiality is expected.

While the practical benefits of DNS pre-resolution are still being debated (e.g., with respect to its effect on cache pollution, the excess load it places on resolvers [3], and the negative impact it may have on performance of applications that do not take advantage of prefetching), this paper attempts to highlight a cautionary tale and hopes to raise awareness that current practices in DNS prefetching could lead to new privacy threats that are ripe for abuse. Specifically, although the adoption of DNS prefetching has already raised specific privacy concerns related to the ability of an inquisitive content author or spammer to monitor the receipt of a well-crafted email [1] or perform timing attacks (e.g., containing customized links [6])[1], we consider how prefetching amplifies disclosure attacks to a degree where it is possible to use cache snooping techniques to infer the likely *search terms* issued by clients behind a particular name server.

The success of these inference attacks relies on the fact that prefetching inserts a significant amount of context into the resolver's cache, allowing the adversary to glean more detailed insights than when this feature is turned off.

To underscore the privacy threats that DNS prefetching can lead to, we examine two distinct modes for enabling disclosure attacks. For simplicity, we first consider the case where the adversary has the luxury of inspecting records from a resolvers' cache—e.g., as might be the case for DNS traffic logs released for research purposes. We then apply the techniques developed for this offline attack to a more realistic remote cache snooping scenario—where the resolver's cache is probed externally in real-time by a remote client. Our primary goal is to raise concern on moving ahead too hastily along this current path, and to stop and think about the potential privacy implications of this design.

The remainder of the paper is outlined as follows. In Section 2 we review related work. Section 3 discusses our goals and outlines the methodology we use. We introduce both offline and online versions of disclosure attacks that are made possible due to aggressive prefetching in Section 4, and discuss their implications as emergent threats. We conclude in Section 5.

2 Related Work

The domain name system plays a critical role in the operation of Internet applications, and so it is not surprising that understanding its performance has been the topic of much research over the past two decades (e.g., [16, 22, 23, 13]). These works all share the common goal of understanding how to improve performance bottlenecks. Jung *et al.* [11] provide extensive analysis of DNS performance and the effectiveness of caching, and also provide a way to model cache hit rates[10].

More recently, several proposals have been suggested for improving the responsiveness of connection establishment by optimistically issuing DNS queries. These ideas include (but are not limited to) prefetching of domain names based on popularity, prefetching of related domain names using piggyback schemes, and precaching of records based on a myriad of renewal policies (see for example, [2, 20, 25]).

More germane to this work is that of DNS cache snooping. Grangeia [8] provides an excellent review of how to remotely inspect a cache for evidence of a *specific* lookup (e.g., `www.nytimes.com`). Remote cache inspection of this type has been used for a number of measurement studies that include, for example, inferring the relative popularity of websites [24, 17] and tracking malware infections [18]. In contrast, in this work we explore how DNS prefetching amplifies new privacy threats, allowing one to gain far more insights than these prior techniques envisioned.

3 Methodology

In order to explore the implications of various browser-based DNS pre-resolution strategies in place today, we designed a framework that allows us to fully automate our data generation process. This framework provides the basic functionality we need to inject keystrokes into several browsers and to automatically collect the resulting DNS data. It is implemented for both Linux and Windows clients, using the `X11` interface and `SendKeys` scripting method, respectively. For the remainder of this paper, we concentrate on Windows clients only, since Windows has the largest user base. To simulate user interaction, our framework accepts a set of terms, actions, and a desired typing rate, and injects keystrokes into a given application. The actions dictate how the framework interacts with the application, for example, whether it enters keystrokes into the location bar or search engine. Our choices for typing speeds are taken from empirical studies [9].

The generation framework is accompanied by a data collection engine which logs all DNS queries and responses created by a single action. Our evaluation was conducted using two disjoint DNS servers, namely, (1) the caching resolver for the computer science department at our institution, and (2) a separate caching resolver that we installed locally. That server forwarded all requests (made by the generation framework) to a public DNS service [7]. We enforce this separation in order to have a control and to avoid polluting the department's caching server. Both servers ran `BIND` version 9. For the analysis in this paper, we collected snapshots of the caches (using `rndc -cache-dump`) at 5 minute intervals for several days in late Februaury, 2010. These "cache dumps" contain no client information, but provide several resource records (`A, CNAME, NS,` etc.) and their remaining times in the name servers' caches [15].

At the time of this study, Internet Explorer only supports DNS prefetching as an extension, and so our examination herein only focuses on Firefox version 3.6 and Chrome version 4.2 for the Windows platform — both of which enable this feature by default.

Prefetching in the Wild

As discussed earlier, optimistic pre-resolution of domain names is implemented as a means to reduce response latency on a "potential" click of a link on a website. How this is realized in the browsers differ, but for the two we consider, they extract the `href` tags from each rendered page, and perform lookups for the resulting do-

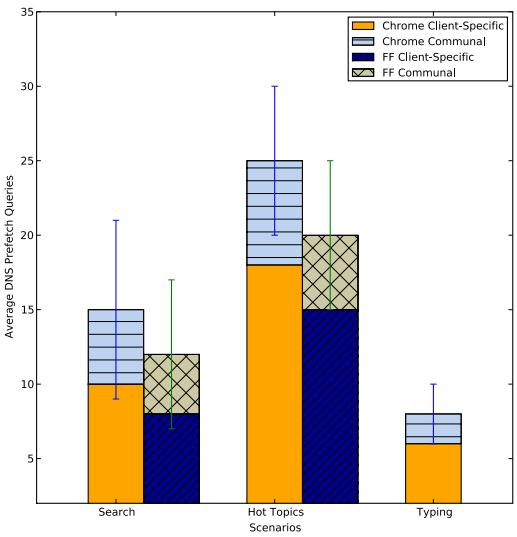

Figure 1: Average DNS request generated.

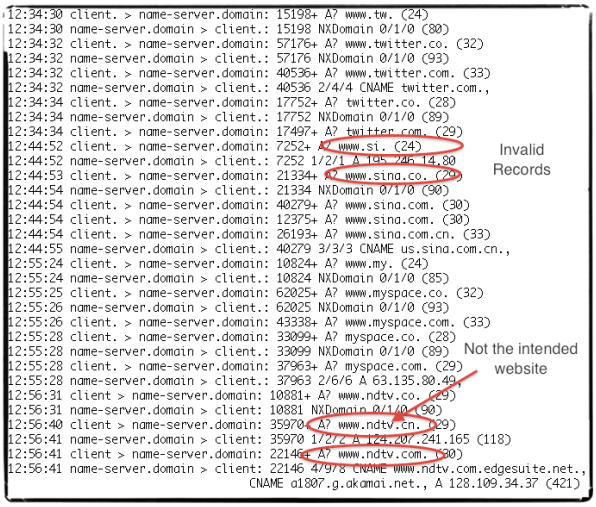

Figure 2: Examples of pre-resolutions as the user types.

mains. Chrome takes this one step further by attempting to guess the site a user might be attempting to visit as she types in the location bar, simultaneously performing pre-resolutions for the predicted destinations.

The fact that these pre-resolutions are occurring will later play a key role in the disclosure attacks we discuss, but for now, we turn our attention to a closer examination of what happens behind the scenes. Recall that each pre-resolution will cause a set of records to be cached at the stub resolvers and their designated upstream full resolvers [21]. Obviously, it would be ideal if prefetching causes the resolvers to cache objects that would benefit other clients as well.

To quickly gauge this "communal benefit", we consider what happens when we 1) search for $n = 40$ distinct keywords, one at a time 2) search for the "hottest" 10 topics (derived from Google Trends) every hour for a 24 hour period and 3) type URLs into the location bar. These three experiments were conducted using the data generation framework, and the target DNS server was the department name server. For each keyword searched, we leave the returned page open for 2 minutes (to simulate a user reading the page), and analyze the request/responses within that interval. Searches on hot topics with Google's search engine generate a dynamic page with real-time scrolling feeds, wherein some pre-resolution is also performed. (This particular search engine optimization has other security-related implications as evidenced by very recent postings; see `http://www.sophos.com/blogs/chetw/g/2010/02/27/tsunami-blackhat-seo-attack/`.)

As a preliminary examination of whether other users could have benefited from the pre-resolutions that occurred because of our searches, we check if a pre-resolved domain was among a daily feed of Alexa's top-100 websites. If so, we consider it as being useful, otherwise the pre-resolution is tagged as client-specific. Then, for each client-specific resolution, we searched a 24 hour period from the departments cache to see if that `A` or `CNAME` record ever showed up again. If it appeared at least once, we take a conservative approach and also count that pre-resolution as being useful (to some arbitrary client).

The results are shown in Figure 1. The histograms are an average of 3 runs for each scenario, with each run happening one day apart. Care was taken to use mutually exclusive searches between the browsers in order to avoid inducing false `TTL` refreshes. Note that a static Google search page usually returns around 10 results, for which the URLs are automatically pre-resolved. However, each result might cause a set of `CNAME` and `A` records to be fetched, especially if they are associated with a content distribution network. Consequently, we see on average of 15 pre-resolutions for Chrome and 12 for Firefox. Moreover, searches for topics that are "hot" cause the browsers to constantly prefetch as new links appear on the page — resulting in a large number of requests during the short 2 minute intervals we kept the search results page opened for.

The pre-resolutions that occur during typed-in navigation are also fairly interesting. Recall that one of the browser's goals is to guess the site the user is trying to visit, providing suggestions along the way to get the user to her destination more quickly. To induce this behavior, we first warmed the browser's history cache by visiting a random set of sites. The server used during this scenario was our control server. Later, we simulated a user typing the URL (sometimes without the `www` prefix) into the lo-

Algorithm 1 Clustering cache entries

Require: DNS Log File
Ensure: Cluster domains in the log into groups based on when they were inserted into the cache

```
anchors = [ ]                          ▷ List of potential anchors
domain_clusters = { }                  ▷ Table of clusters
Alexa = { }                            ▷ Alexa's daily top-100
for all DNS Records in Log do
    if Record.domain not in Alexa then
        authoritative_ttl = GetSOA(domain)
        time_in_cache = authoritative_ttl - domain.ttl
        anchors.Append(<domain, time_in_cache>)
    end if
end for
while anchor in anchors do
    ▷ Time elapsed since anchor was added to cache
    age = anchor.time_in_cache
    for all a in anchors do
        if a.time_in_cache == age ± window then
            domain_clusters[window].Append(a)
            delete a from anchors[]
        end if
    end for
end while
```

Algorithm 2 Keyword Extraction

Require: Domain Name, Prefix Size
Ensure: Tokenized List of Words

```
words = [ ]                            ▷ List of extracted words
word_trie = Trie()
current_word = domain
char_consumed = 0
m = Prefix Size
while char_consumed < length(domain) do
    for all keyword in suggest(current_word[:m]) do
        word_trie.add(keyword)
    end for
    if match = word_trie.find-prefix(domain) then
        words.append(match)
        current_word = domain - match
        char_consumed += length(match) - m
    else
        m = m+1
        char_consumed += m
    end if
end while
```

cation bar. As the browser attempts to guess the user's intention, DNS queries are created; most times after only a few characters are typed.

Figure 2 provides a brief illustration of this behavior. Notice that most of the resulting prefetches result in `NX` responses, or even valid domains for sites where the user had no intention of visiting (e.g., `www.ndtv.cn`). The result in Figure 1 is the average number of pre-resolutions across 20 typed-in entries. In this case, over 95% of these resolutions resulted in `NX` responses.

While the issue of whether these prefetching mechanisms do more harm than good is debatable (e.g., the real-time searches could cause an increase in cache evictions due to the increased rate of "client-specific" DNS records), one thing is for sure — the additional queries provide context which can be used to facilitate emergent privacy threats.

4 Disclosure Attacks

In what follows, we consider how prefetching can be abused by an adversary in order to reconstruct searches by clients served by a particular DNS server. Before diving into the specifics of these attacks, lets first recall how we typically find information on the Web today. Generally speaking, we input a set of keywords (e.g., *"stem cell controversy"*) into our favorite search engine, and then explore the ranked set of returned links (clicking on, say, the top three results). As we refine our search term, the more advanced engines provide suggestions on terms that could yield better results (e.g., *"stem cell research debate"*). These suggestions are created using item-based or user-neighborhood based recommender algorithms [5, 12]. The key here is that for these suggested terms, the set of links pre-resolved for the resulting search results will be relatively stable.

4.1 Offline Attack

The first inference attack assumes access to logs (`BIND` cache dumps in this case) and attempts to reconstruct the searched terms. The challenges here are in first grouping "related" domain names in this log, tokenizing the domains in order to extract keywords, and using a n-recommender algorithm to build queries based on the extracted keywords. In what follows, we discuss each of these challenges in turn.

Clustering of Entries Recall that pre-resolving domain names on a search result pages results in a set of simultaneous DNS queries. The responses to these queries are cached along with their TTL values. Since these queries are issued in close succession of each other, they would age at the same rate in the cache. Hence, it is possible to group related domains by comparing the current TTL in cache with the authoritative TTL, thereby computing the age of each record. The assertion is that records with the same age are likely to have been fetched because of pre-resolutions.

Actual Query	First guess	Second guess	Third guess
"Gambling Addiction"	gambling addiction	gambling age	addict
"Alcohol Withdrawal Syndrome"	alcohol withdrawal symptoms	alcoholics anonymous	alcohol poisoning
"Gun Control"	gunbroker	guns for sale	⟨none⟩
"Racism In America"	racism america	racism today	racism facts
"Biological Weapons"	biological warfare	weapons	⟨none⟩
"Homelessness In America"	homelessness america	homelessness statistics	homelessness facts
"Immigration Reform"	immigration naturalization	immigration illegal	immigration news
"Human Cloning"	cloning humans	cloning	cloning animals
"Internet Privacy"	internet privacy	internet crime	internet explorer
"Domestic Violence"	domestic violence	domestications	domestic abuse

Table 1: Top three guesses for 10 different queries.

Our clustering approach (given in Algorithm 1) is straightforward. The basic idea is to create a list of "anchors" during an initialization phase and then group domains with similar age. We also fetch the authoritative TTL's for each domain in the list.

At first, we assume all domains are anchors[2]. Next, a domain (usually the first record) from the anchor list is chosen and its age is computed (i.e., authoritative TTL - current TTL). We then sequentially scan the subsequent elements in the list for domains with the same age ($\pm window$), where the window is a tunable parameter. Finally, elements with the same age are considered as a cluster. Each iteration over this list removes anchors as they become members of a cluster. Table 2 shows the derived grouping for the query "*steroids in baseball*"; noticed that all the domain have the same approximate age (600 seconds).

Domain Name	Auth. TTL	Current TTL	Age
teenink.com	3600	3001	599
rcshield.com	3600	3000	600
steroid.com	10800	10198	602
steroidsinbaseball.net	14400	13802	598
baseballssteroidera.com	14400	13800	600

Table 2: Example cluster showing pre-resolved domains when searching for "*steroids in baseball*".

Keyword Extraction Once the entries have been clustered, tokenization begins (see Algorithm 2). Our approach leverages an n-suggest algorithm to obtain possible words for a given prefix (the first m-characters). The possible matches are fed into a Trie, allowing us to perform longest prefix matches on the domain name. The algorithm then iterates over the next set of characters and the process repeats until all the characters are consumed and tokenized.

The output of Algorithm 2 is a list of words ordered from left to right for each domain name, j (i.e., we have $w_{j1}, w_{j2}, \ldots, w_{jk}$). Table 3 depicts an example tokenized list for each domain for the cluster based on the aforementioned query. We then take all the first-order keywords (i.e., w_{j1}) and rank them by frequency. In the previous example, "steroids" has the highest rank, followed by "baseball", etc. Next, we again take advantage of an n-recommender systems, and construct a search using all first order words with frequency $> \delta$. At this point, we have a list of suggestions. Each suggestion is compared with our list of ordered words, and we output (as our guess) the suggestion with the maximum number of matches. The final ranking is computed using the weighted frequency of all the words in the inferred query.

Domain Name	Keyword List
teenink.com	teen, ink
rcshield.com	shield
steroid.com	steroid
steroidsinbaseball.net	steroids, baseball, in
baseballssteroidera.com	baseballs. steroids, era

Table 3: Example extracted keywords for pre-resolved domains for "*steroids in baseball*".

4.1.1 Preliminary Results

We evaluated the outlined approach using the cache dumps from the departmental server. The data generation framework was used to inject 50 search queries at random intervals over a day. We used both browsers in this test. The task at hand was to predict what where the likely queries by inspecting the cache dumps. The results for a handful of these inferences are shown in Table 1. The table shows the actual query injected and the top three inferred searches for each query. Notice how strikingly similar they are.

As a preliminary assessment of the accuracy of the outlined approach, we computed true and false positive rates based on obtaining snapshots of the server's cache

Granularity (mins)	FP %	TP %
5	3%	85%
10	4.5%	82.5%
15	6%	78.5%
30	9%	74%
60	14.5%	72.5%

Table 4: Conservative estimate of the accuracy of reconstruction assuming cache dumps of varying granularity.

at different granularity. The set of words in the original query is defined as Q_w and the set of words in the result is R_w. A *true positive* and *false positive* for R_w is computed as:

$$TP = \begin{cases} \frac{|R_w \cap Q_w|}{|Q_w|} & \text{if } R_w \subsetneq Q_w \\ 1.0 & \text{if } R_w = Q_w \\ 0.0 & \text{if } Q_w \subsetneq R_w \end{cases}$$

$$FP = \begin{cases} \frac{|R_w \setminus Q_w|}{|Q_w|} & \text{if } Q_w \subsetneq R_w \\ 1.0 & \text{if } R_w \neq Q_w \\ 0.0 & \text{if } R_w \subsetneq Q_w \end{cases}$$

Clearly, this reflects a conservative computation for the true positives, as we only count proper subsets and complete matches as hits. Likewise, any guess that contains even a single word that is not present in the original query is counted as a false positive. Arguably, the results would be significantly improved if we considered the semantic advantage of combining words in the top-3 guesses based on feedback from a human observer. The results for all 50 search queries over a four hour window are shown in Table 4. As expected, smaller granularity in the snapshots yields better accuracy. The high true positive rate of over 70% even for relatively large granularity (of 60 mins) is due to the presence of CNAMEs whose TTL values are often in hours. The increase in false positive rate is caused by incorrectly clustering of domains with the same age, but which were caused by different prefetching events. The false positive rate is also influenced by the nature of a search term, highly popular searches that are general in nature (e.g. Hot Topics) cause an increase in the false positive rate, whereas specific searches (e.g. Steroids in Baseball) yield a lower false positive rate.

4.2 Remote Cache Inspection

The attack discussed previously assumes access to cache traces or DNS logs, which arguably, may not be a very practical assumption. That said, the observation that the related domain names prefetched for a given search term will have a similar age in the cache, can be used to construct an online probing attack. In this case, the adversary

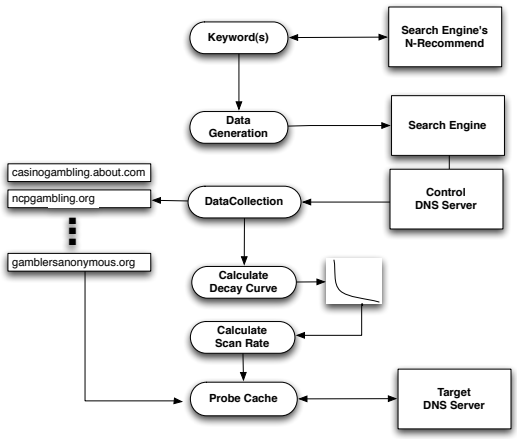

Figure 3: Control flow for online attack

is interested in knowing if some set of searches were performed by clients of a target name server.

Figure 3 outlines one way the adversary could carry out such an inference. Given as set of keywords of interest, the first step is to create a profile, $\mathcal{P}$, for the keywords. A profile simply contains the set of domain names that would be prefetched using this search term, along with the corresponding authoritative TTLs for each pre-resolved name. These TTL values are used to create a decay curve (as shown in Figure 4), which models the percentage of items in $\mathcal{P}$ that would be present in cache after some predefined amount of time has elapsed beyond a client's search for that term. Using this information, the attacker picks the desired accuracy threshold she is willing to tolerate, and notes the corresponding age value, t. Let $\mathcal{D}$ be the set of domains in $\mathcal{P}$ that have an age less than t. Additionally, set the probing rate, r, $< t$.

Next, the adversary selects a domain name, $d_i \in \mathcal{D}$ (e.g., the one with the mean age value), and uses cache snooping techniques [8] to inspect the target for d_i. If she receives a cache hit, then she immediately tests for the presence of the other elements in $\mathcal{D} \setminus d_i$; otherwise, she continues to inspect the cache at the probe rate of r.

When a cache hit occurs, the attacker can compute the amount of time this entry has been residing in the cache as before. She does so for all the domains in $\mathcal{D}$ that were cache hits. All the hits with the same age ($\pm$ few seconds) are counted as a match on the profile. Intuitively these are domains that were added to the cache at the same time, most likely because of the browser's DNS prefetching event. Finally, she computes her success rate by calculating the percentage of matches received. A high percentage of matches allows her to conclude that the target search query was performed by a client of the target name server.

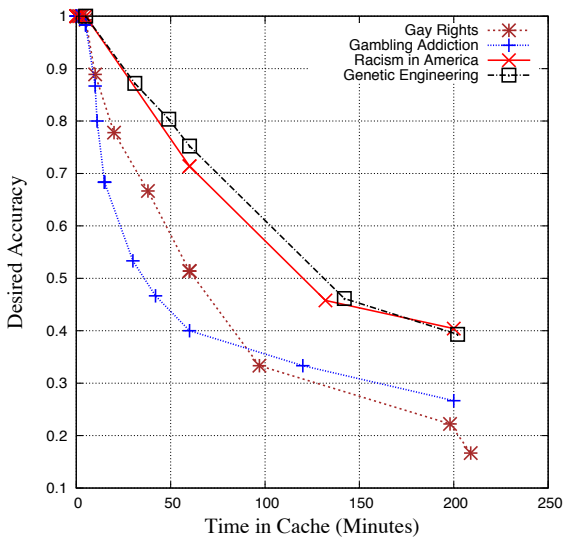

Figure 4: Sample decay curves for four search terms

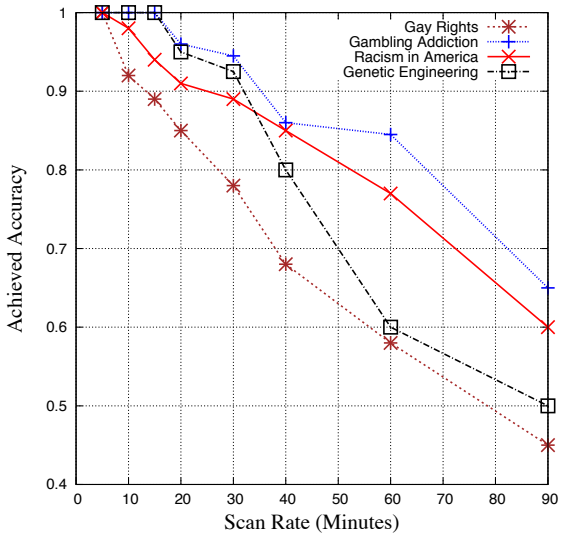

Figure 5: Accuracy of the online cache snooping attack for four search terms of interest.

4.2.1 Preliminary Results

To evaluate the effectiveness of this attack, 10 profiles of interest, $\mathcal{P}_1, \ldots, \mathcal{P}_{10}$ were built using the data generation and collection framework with the control resolver. A decay curve was built for each one of the profiles, and the respective probing rates, r_i, were set at the value corresponding to a desired accuracy threshold of .75. For the four examples shown in Figure 4 notice that it is possible to periodically probe the cache every 30 mins and still achieve a good hit rate.

We then used our data generation framework to perform a set of searches at random times during a 4 hour window. No search was performed more than once. The Chrome browser was used in this test, and the resolver was set to the departmental name server. During that period, we snooped the cache of the departmental name server at the inferred rates. Figure 5 shows the result of the attack. Accuracy in this plot is defined as the percentage of the search term's profile we received cache hits on. The results show an average of over 90% accuracy with a scan rate of 10 minutes, and 85% accuracy at a conservative scan rate of 30 minutes. For some searches, the accuracy is reasonably high even with scan rates as low as every hour primarily because `CNAME` records tend to remain in cache for a long time.

We also considered the success of our approach (for the same search queries as before) when prefetching is turned off. In order for the attack to succeed in this case, the client must first click on a link from the search results; otherwise the resolution would not appear in the target's cache. In lieu of any empirical results that shed light on the probability of clicking on a link, we approximated the click probability as follows: we assumed that the probability that a user clicks on any of the top three links is high ($.2, .15, .15$, respectively). The remaining probability is uniformly distributed amongst the other links.

The success rate of a cache-snooping attack is calculated based on the click probability *and* the requirement that tokens from the clicked domains contain contextual information about the search itself. For example, `en.wikipedia.org` would contain no specific information about a search for *"single malt scotch"*, but `www.scotchwhisky.net` would. The median value for the success rate when prefetching is disabled is 5% for a scan rate of 5 minutes, compared to 88% when prefetching is turned on.

Obviously, the inferences made only shed light on the searches being performed by the population of clients (as a whole) that use the resolver the adversary is probing. Therefore, if the server is used by a very diverse populations of clients, then one can not tie these searches to a particular organizational unit (e.g., client of UNC's CS department). Hence, a reasonable approach for savvy clients that are concerned about the attacks outlined herein might be to use a public DNS service to achieve some level of anonymity.

5 Summary

Obviously, the inference attacks we outlined depend on accurately computing the age of records in the cache. However, it is possible that a target server may not obey the authoritative TTL when caching an entry. `BIND-9`, for example, lets server administrators set a maximum

cache TTL value. In such cases, we could incorrectly compute the age of items in the cache, leading to poor predictions. To limit this issue, one could use the sanitization techniques explored elsewhere [18, 4] to first check if the target name server abides to authoritative TTL values. Performing this check does require that the target resolver be an open resolver, but that does not appear to be a significant issue in practice; for example, a 2009 DNS survey [14] estimates that there are as many as 13 million open resolvers on the Internet.

Another practical limitation of our current approach is that the search profiles should be stable for the entire time period of the probe activity. Likewise, our use of a n-recommendation algorithm for tokenization in the offline attack does come with caveats. For instance, if the domain names contain no identifiable words or none of the tokenized words adequately match the search term, false negatives will occur. Nonetheless, we believe the issues raised in this paper serve to shed light on practices we may want to rethink going forward. In particular, our ability to reconstruct search queries when prefetching is turned on underscores the thin line we walk between increased Internet browsing speed and privacy.

Our main objective in this work is to highlight the fact that if left unchecked, rapid enhancements in *when* and *how* DNS prefetching is performed could lead to new security and privacy threats. Thankfully, as of this writing, both Firefox and Chrome provide users with mechanisms to turn off DNS prefetching—the specifics of which are provided in Appendix A. We hope that in future browser updates DNS prefetching is turned off by default, or at the very least, the developers make it easier to disable this feature.

6 Acknowledgement

We thank the anonymous reviewers for their insightful comments. This work was supported in part by the National Science Foundation under award number 0831245. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF.

A Disabling Prefetching

For Chrome users, DNS prefetching can be disabled by unmarking the check box "use DNS prefetching to improve page load performance" via the $Tools \rightarrow Options \rightarrow Under\ the\ Hood$ sub-menu. For Firefox, disabling this feature is less obvious. Users can do so by setting the `network.dns.disablePrefetch` preference to true using the about:config method. For some versions of Firefox, it appears that the `network.dns.disablePrefetchFromHTTPS` preference should also be set it to true in order to fully disable DNS prefetching. Similarly, for other Mozilla Necko-based apps (like Thunderbird), these preferences can be set by editing the `user.js` file in the user's profile folder.

References

[1] CHROME TEAM. The Chromium Projects. See:`http://www.chromium.org/developers/design-documents/dns-prefetching`.

[2] COHEN, E., AND KAPLAN, H. Proactive Caching of DNS Records: Addressing a Performance Bottleneck. In *Proceedings of the IEEE Symposium on Applications and the Internet* (2001), pp. 85–94.

[3] DAGON, D. DNS Security: Lessons Learned and the Road Ahead. Invited Talk, USENIX Security Symposium, Aug 2009.

[4] DAGON, D., LEE, C., LEE, W., AND PROVOS, N. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of the* 15^{th} *Network and Distributed Systems Security Symposium* (2008).

[5] DESHPANDE, M., AND KARYPIS, G. Item-based Top-N Recommendation Algorithms. *ACM Transactions on Information Systems 22*, 1 (2004), 143–177.

[6] FELTEN, E. W., AND SCHNEIDER, M. A. Timing Attacks on Web Privacy. In *ACM Conference on Computer and Communications Security* (2000), pp. 25–32.

[7] GOOGLE ENGINEERS. Introduction to Google Public DNS. See `http://code.google.com/speed/public-dns/docs/intro.html`, Dec. 2009.

[8] GRANGEIA, L. DNS Cache Snooping or Snooping the Cache for Fun and Profit, Feb. 2004.

[9] JAY, C., GLENCROSS, M., AND HUBBOLD, R. Modeling the Effects of Delayed Haptic and Visual Feedback in a Collaborative Virtual Environment. *ACM Transactions on Computer-Human Interaction 14*, 2 (2007), 8.

[10] JUNG, J., BERGER, A. W., AND BALAKRISHNAN, H. Modeling TTL-based Internet Caches. In *IEEE Infocom 2003* (April 2003).

[11] JUNG, J., SIT, E., BALAKRISHNAN, H., AND MORRIS, R. DNS Performance and the Effectiveness of Caching. *IEEE/ACM Transactions on Networking 10*, 5 (2002), 589–603.

[12] KARYPIS, G. Evaluation of Item-Based Top-N Recommendation Algorithms. In *Proceedings of the 10^{th} International Conference on Information and Knowledge Management* (2001), pp. 247–254.

[13] LISTON, R., SRINIVASAN, S., AND ZEGURA, E. Diversity in DNS Performance Measures. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), pp. 19–31.

[14] MEASUREMENT FACTORY. DNS Survey. See `http://dns.measurement-factory.com/surveys/200910.html`, Oct. 2009.

[15] MOCKAPETRIS, P. V. Domain Names - Concepts and Facilities, 1987.

[16] PAXSON, V., AND FLOYD, S. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking 3*, 3 (1995), 226–244.

[17] RAJAB, M. A., MONROSE, F., TERZIS, A., AND PROVOS, N. Peeking Through the Cloud: DNS-Based Estimation and Its Applications. In *Applied Cryptography and Network Security Conference* (2008), pp. 21–38.

[18] RAJAB, M. A., ZARFOSS, J., MONROSE, F., AND TERZIS, A. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC)* (Oct., 2006), pp. 41–52.

[19] SECURITY FOCUS. CVE-2010-0464: Multiple Vendors Email Clients DNS Prefetch Information Disclosure Vulnerability. See `http://www.securityfocus.com/bid/38046`, Feb. 2, 2010.

[20] SHANG, H., AND WILLS, C. E. Piggybacking Related Domain Names to Improve DNS Performance. *Computing Networking 50*, 11 (2006), 1733–1748.

[21] VIXIE, P. DNS Complexity. In *ACM Queue* (May 2007).

[22] WESSELS, D. Is Your Caching Resolver Polluting the Internet? In *Proceedings of the ACM SIGCOMM workshop on Network troubleshooting* (2004), pp. 271–276.

[23] WESSELS, D., AND FOMENKOV, M. Wow, That's a Lot of Packets. In *Passive and Active Measurement Workshop* (April 2003).

[24] WILLS, C. E., MIKHAILOV, M., AND SHANG, H. Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), pp. 78–90.

[25] ZHANG, Z., ZHANG, L., EN XIE, D., XU, H., AND HU, H. A Novel DNS Accelerator Design and Implementation. In *APNOMS* (2009), pp. 458–461.

Notes

[1] Indeed, several prefetching-related CERT advisories were recently released about such vulnerabilities [19].

[2] The list is pruned first by omitting the Alexa top-100 domains (e.g. wikipedia , twitter, etc.) as they could be shared by many clusters.