Inflight Modifications of Content: Who are the Culprits?

Chao Zhang
Polytechnic Institute of NYU

Cheng Huang
Microsoft Research

Keith W. Ross
Polytechnic Institute of NYU

David A. Maltz
Microsoft Research

Jin Li
Microsoft Research

Abstract

When a user requests content from a cloud service provider, sometimes the content sent by the provider is modified *inflight* by third-party entities. To our knowledge, there is no comprehensive study that examines the *extent* and primary *root causes* of the content modification problem. We design a lightweight experiment and instrument a vast number of clients in the wild to make two additional DNS queries every day. We identify candidate rogue servers and develop a measurement methodology to determine, for each candidate rogue server, whether the server is performing inflight modifications or not. In total, we discover 349 servers as malicious, that is, as modifying content inflight, and more than 1.9% of all US clients are affected by these malicious servers. We investigate the root causes of the problem. We identify 9 ISPs, whose clients are predominately affected. *We find that the root cause is not sophisticated transparent in-network services, but instead local DNS servers in the problematic ISPs.*

1 Introduction

Online advertising has been revolutionizing the advertisement business for many years. As one of the fastest growing Internet businesses, online advertising is highly lucrative and profitable, so much so that even giant companies (e.g., Google) with tens of thousands of employees and tens of billions of revenue can build their entire business around online advertising. However, also in this space, are rogue companies that secretly "steal" away advertisement revenue from existing providers.

Indeed, as [14] has demonstrated, such malicious practices do exist. When users access content services, content delivered from the service providers to the users can be modified *inflight*. Such modifications can change the content itself, the embedded advertisements, or even redirect users to undesirable destinations. However, although [14] demonstrates evidence of such malicious practices, to our knowledge there is no comprehensive study that examines the extent and the primary root causes of the modification problem. A cloud service provider (such as Facebook, Google, Microsoft and Yahoo!) would certainly want to know: (i) What fraction of its users are subject to inflight modifications? (ii) What types of modifications are usually taking place?

More importantly, cloud service providers would like to identify the root causes of the inflight modifications, so that they can take appropriate measures to defend their businesses. For example, if evidence shows that the modifications are carried out by Internet Services Providers (ISPs), then complaints can be made to regulatory agencies or legal actions can be taken. On the other hand, if evidence shows that the modifications are due to the users being affected by malware, then alerting the users and offering solutions to combat the malicious software are more appropriate. Existing studies [14, 16], unfortunately, do *not* provide methodologies for such classification. Therefore, new methodologies need to be developed to identify the root causes. In summary, the real challenge is to develop a methodology that not only detects the existence of inflight modifications, but also identifies the root causes of such modifications. Such a methodology can eventually help cloud service providers defend their online advertisement business.

To this end, we make the following contributions:

- We design a lightweight experiment and instrument a vast number of clients in the wild to make two additional DNS queries every day. By aggregating the data collected from over 15 million clients, we identify 4,437 candidate rogue servers.

- We develop a measurement methodology to determine, for each candidate rogue server, whether the server is performing inflight modifications or whether it is benign. Among the 4,437 candidate rogue servers, 349 are deemed malicious, that is, are modifying content inflight. Astonishingly, more than 1.9% of all US clients are affected by these malicious servers!

- We investigate the root causes of the problem. We identify 9 ISPs, whose clients are predominately affected. We find that the root cause is not sophisticated transparent in-network services, but instead local DNS servers in the problematic ISPs.

2 Data Collection

In this section, we develop an experiment that a cloud provider (say `www.example.com`) can use to determine the servers that are candidates for making inflight modifications of its content. Additionally, the experiment determines (i) for each candidate server, the IP addresses of the clients that are obtaining content from the server, and (ii) for each such client in the wild, the IP address of the Local DNS (LDNS) server it is using.

Our data collection experiment instruments a vast number of clients in the wild, each of which makes lightweight measurements. Specifically, each instrumented client infrequently resolves `www.example.com`, then reports to our data collection server three pieces of information: (i) the IP address of the client; (ii) the IP address of the LDNS used by the client to resolve `www.example.com`; and (iii) the IP addresses of the servers returned by the LDNS when resolving `www.example.com`. These servers returned by the LDNSes become our "candidate rogue servers". Some of these candidate servers will be legitimate servers operated by `www.example.com` or one of its partners; other candidate servers will actually be rogue servers that perform inflight modifications.

To collect this data, each instrumented client once everyday resolves the hostname `www.example.com`. During this DNS resolution process, the client sends a DNS query to a local DNS server (LDNS), which is usually (but not always) operated by the client's ISP. The LDNS interacts with the DNS system, and returns the answer back to the client. The instrumented client then sends to our data collection server its IP address, the IP address of its LDNS server, and the IP addresses of the servers returned by its LDNS.

2.1 Collecting the LDNS Addresses

As described above, we want the instrumented client to report the IP address of its LDNS (as well as its own IP address and the server IP addresses). One possible way to obtain the LDNS IP address is to get it directly from the client's OS. However, depending on how the client is instrumented, such direct access to the LDNS IP address may not be possible. We now describe a lightweight experiment, called *DNS Echo*, which can be implemented by most instrumentation approaches. This approach is similar to the Java Applet in Netalyzr [13].

A special hostname `echo.example.com` is created for the experiment. The authoritative name server for the domain `echo.example.com` is instructed to respond to any DNS query for `echo.example.com` with the source IP address of the query. Whenever an LDNS server queries the authoritative name server for `echo.example.com`, it obtains an answer containing the IP address of itself. This IP address (of the LDNS server) is then returned to the client that initiated the DNS query. The process is illustrated in Figure 1.

One might question whether a rogue LDNS server would simply return the IP address of a rogue web server to the query of `echo.example.com`. Our experience shows that rogue LDNS servers only interfere with the DNS resolution of a few selected domains, and they handle all other domains in the normal manner.

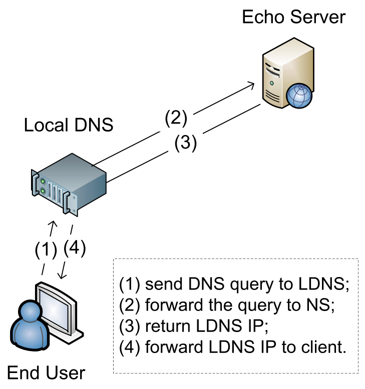
Echo Server
Local DNS
(2)
(3)
(1)
(4)
End User
(1) send DNS query to LDNS;
(2) forward the query to NS;
(3) return LDNS IP;
(4) forward LDNS IP to client.

Figure 1: Collecting LDNS with DNS Echo Experiment

2.2 Instrumenting Clients

The instrumented clients in the wild need to resolve two hostnames – `www.example.com` and `echo.example.com` – and report the obtained IP addresses, as well as the client's own IP address, to our data collection server. This can be done in many different ways. For example, one could embed a Java Applet with such functionality in popular web pages. When a client visits the web pages of `www.example.com`, its browser will load and execute the Java Applet, which then resolves the hostnames and reports the IP addresses to our data collection server. Alternatively, the functionality can be integrated into popular software that is distributed to worldwide clients.

For our experiment, we added this functionality to an optional piece of software downloaded and used by millions of users. A certain percentage of software periodically execute the experiment and report the obtained IP addresses to our data collection server. We emphasize that the clients do *not* establish any connections with the obtained IP addresses – they merely perform two DNS queries and report the IP addresses. Each DNS query is a few tens of bytes, so the experiment generates minimal additional traffic for each client. The report from a client does reveal the client's IP address (and no other information that could identify the client). However, the very same address is already available when the client downloads the software. Therefore, our experiment does *not* reveal any additional identifying information about the clients. In addition, data is collected only from the users that have opted in to share with Microsoft data that will help improve their experience, and it is completely anonymized after 6 months in accordance with Microsoft's standard data privacy policies.

The experiment was repeated by each client once per day and lasted for two months, between Sep. 1 and Oct. 31, 2010.

3 Identifying the Rogue Servers

We investigate inflight modifications for a popular Internet search service, which we refer to as `www.example.com`. Through the instrumentation described in the previous section, we collected the IP addresses for `www.example.com` that were resolved by millions of clients in the wild. This list is surprisingly large – there are 4,437 unique IP addresses. Each of these addresses is a candidate for a rogue server that performs inflight modification. To determine which of these servers are indeed rogue servers, we develop the Revealer Platform, as described below.

3.1 The Revealer Platform

Not all of 4,437 servers in the list are problematic. For example, a simple web proxy would be included in the list, even though it does not modify content at all. In this section, we describe a semi-automatic framework, called *Revealer*, which probes individual servers in the list and identifies the truly problematic ones.

The key idea is, for each candidate server, to access the content through each candidate server and also directly from `www.example.com`. The server is *benign* if the content is the same. On the other hand, the server is *malicious* if it produces different content, such as inserting or modifying advertisements.

With scalability and repeatability in mind, we develop Revealer to identify malicious servers. Revealer is scripted using the Chickenfoot browser automation framework [5] and employs a semi-automatic verification procedure. Chickenfoot works as a extension of Firefox, so Revealer uses the Firefox browser.

Revealer consists of three components: a controller and two Chickenfoot-based script instances. The controller iterates through a list of search query URLs and assigns them one by one to both instances. One instance, called *Server Prober*, retrieves the URL from the candidate server. The other, called *Legit Server Prober*, retrieves from a corresponding legitimate server. The two instances run on two different machines with the same configuration and are synchronized using a shared file lock. The content retrieved through both probers is compared to determine whether the candidate server is malicious. The flow is illustrated in Figure 2.

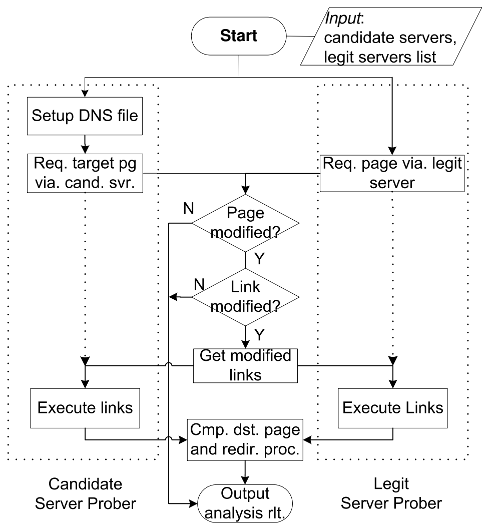

Figure 2: The Revealer framework: Controller, Server Prober and Legit Server Prober.

To force the browser to access the service through the candidate server, Server Prober first modifies the local DNS resolution file so that `www.example.com` points to the candidate server. [1]. Then, it starts a tshark [4] packet sniffer to capture all incoming and outgoing HTTP traffic. Finally, Server Prober starts Firefox, which loads the Chickenfoot scripts to retrieve a specified URL. The retrieved content is saved to local storage for further analysis. Legit Server Prober follows a similar procedure to retrieve content from a legitimate server.

The controller compares the files fetched through the candidate server and from the legit server. If a text string is changed, the candidate server is recorded as problematic. If link-related change is detected, we need to do more tests. Because our cloud service is an Internet search service, the search results are often accompanied with ads. Therefore, given the same query, when fetching two pages at the same time from the same host, the search result pages are possibly different if different ads are generated by the advertising system. To deal with this case, when link-related changes are detected, the different links are sent to the Legit Server Prober and executed in the Firefox browser. If these links are not recognized due to the modification which happened on the path to the client, the legitimate search server will show the error "web page doesn't exist". For example, a modified link, `http://www.bing.com/aff?p=JZLJk***`, cannot be served by Bing servers.

At the Candidate Server Prober side, for the links different with the ones from the Legit Server Prober, Revealer follows the URLs embedded in the web page, as if users are clicking the corresponding links in the page.

[1] On Windows platform, the DNS file is `~\WINDOWS\system32\drivers\etc\hosts`; on Linux, it is `/etc/hosts`

The Chickenfoot cannot directly access the HTML document. Instead, it can only access the objects, known as internal Document Object Tree (DOM), from rendered HTML document. Therefore, Chickenfoot locates the embedded URLs in the DOM through string matching and loads these URLs to emulate users clicking the links. Importantly, the click operation may trigger the execution of JavaScript functions inserted by the candidate server if there is any. So, following the links helps us detect such insertions, while simply fetching the web page for the specified search URL does *not* trigger these Javascript functions.

The two files retrieved through the candidate server and the legitimate server are compared to identify modifications. Moreover, any referenced pages generated by the two servers are also compared. If there is a difference, then the candidate server has performed inflight modification. Each detected malicious server is outputted to a text file, and then removed from the list. Revealer keeps testing the servers on the list in a round-robin fashion, since malicious servers may randomly choose to modify pages.

3.2 Types of Modifications

Next, we elaborate various types of modifications discovered by Revealer. Some modifications are quite obvious and easy to catch by ordinary users, while others are very stealthy and are almost impossible to discover by untrained users.

Modify search result links. In a search result web page, the result links are simply replaced. For example, when searching "dell computers," the correct result page contains a link pointing to an entry about Dell on Wikipedia. However, we have observed a malicious server that returns a page for which the text of the link still refers to Wikipedia, but the link is now changed to `http://www.example.-com/goto?id=5d***`. If a user clicks the modified link, she will be directed to a third party web site, instead of Wikipedia. This type of modification is easy to identify.

Note that the modified link points to `www.-example.com`, instead of the third party website. Such modification ensures, when the user clicks the modified link, she will again connect to the malicious server, which then redirects the user to an arbitrary third party web site. We conjecture the purpose is to make inflight modification flexible.

Modify advertisement links. Advertisement link is another common type of modification. For example, a correct advertisement link `http://www.-example.com/?ld=***` is replaced by `http://-www.example.com/aff?p=***`. If the user clicks the link, she will end up visiting a different advertisement than the original one. The only change here is the request parameter after the hostname. Ordinary users typically do *not* notice such modification. Yet, it can be easily detected once the links obtained from the candidate server is compared to the correct ones from a legitimate server.

Insert JavaScript. A malicious server might *not* modify either search result links or advertisement links, but rather insert a piece of JavaScript code into the result page, which then modifies the links only when the user clicks them. When the modified page is displayed in the user's browser, all the result links and advertisement links appear normal (when the user moves the mouse over the links). Once a link is clicked, however, the JavaScript code is invoked with the original link as a parameter. The JavaScript code returns a new link, which ultimately loads a completely different web page. Such a modification is inconspicuous – it cannot be identified even by comparing all the links. It can only be discovered if the browser follows each link in the result page and saves the retrieved content from the links.

Redirect requests. In this case, a malicious server does *not* modify the result or advertisement links; instead, it redirects the query. We discovered two types of redirection. The first type redirects a search query to a different search engine. The second type inserts several rounds of redirection before eventually directing the user to her destination.

It is instructive to understand the economic motivations behind this second type of redirection. After carefully examining the inserted links, we discover that they are all related to several online advertisement companies. These companies get paid when their advertisement links are clicked by users. The extra rounds of inserted redirection are used to generate clicks, as if they are from a large number of real users.

In addition, we noticed that this type of modification is extremely stealthy. It only intercepts and redirects if the search queries are generated from the *address bar* of web browsers. Clearly, only intercepting the queries from the address bar reduces the risk of exposing the malicious servers. Interestingly, most malicious servers that we have discovered belong to this category.

Aggressive modifications. We also observed two servers that aggressively change the result web pages. One server inserts banner advertisements in the home page of the search service, while the other replace the the result pages with totally different contents, links, and ads.

Table 1 gives a summary of the types of malicious servers discovered. We discovered a total of 349 malicious servers. Of 154 redirected requests from the address bar and 72 inserted Javascript.

4 Which Clients are Affected?

We now investigate the client population affected by the identified malicious servers. Our dataset contains

Type of proxy	# of IP
Modify search result links	41
Modify ad links	80
Javascript injection	72
Redirect requests from address bar	154
Modify whole search results	1
Inject ads on homepage	1

Table 1: The types of malicious servers.

15,688,909 unique clients worldwide. Among them, about 0.9% (137,871) were (at least once) directed to one of the 349 malicious servers identified by Revealer. *The percentage is even more significant among the clients in the US, reaching an astonishing 1.9%!*

4.1 Location Breakdown

We first investigate how the affected clients distribute geographically. To this end, we look up the 137,871 affected client IP address in the Quova IP GeoLocation database. We find that over 95.4% affected clients are in the US. All other countries, with the exception of Haiti, have a percentage below 0.17%. Thus we see that the bulk of this attack is being directed to US clients.

To gain more insight into how wide the affected clients spread geographically, Table 2 lists the number of affected clients by cities. Interestingly, all the cities appearing at the top of the list are located in US. Moreover, some US cities are heavily affected whereas others are hardly affected. Many of these top cities have relatively small populations; many large cities – including New York, Los Angeles, and Houston – do not appear on the list.

Rank	City	# of affected	# of total	%
1	Germantown	23,822	30,614	77.8
2	Troy	8,179	13,771	59.4
3	Cincinnati	6,071	36,462	16.7
4	Ashburn	4,465	62,945	7.1
5	Columbus	3,929	37,043	10.6
6	Naperville	3,782	5,206	72.6
7	Rochester	2,556	24,115	10.6
8	Richmond	1,805	18,968	9.5
9	Clarks Summit	1,676	2,076	80.7
10	Elk Grove	1,500	2,074	72.3

Table 2: Top 10 cities ranked by the number of affected clients.

4.2 ISP Breakdown

We also obtain the Autonomous System Number (ASN) of each client, affected by the malicious servers, by looking up the client IP address in the Quova IP GeoLocation database. We map the ASN to its ISP using the table from [3] and aggregate all the clients within the same ISP together. In total, there are 1,549 ISPs affected. We remove the ISPs with less than 50 affected clients, which reduces the number of affected ISPs to 79. After this filtering, the total number of affected clients is 131,358. Table 3 shows the top 20 ISPs ranked by percentage of affected clients. We see that some ISPs are heavily affected whereas the majority of the ISPs are hardly affected. For example, the first 9 ISPs, more than 65% of the clients are affected, whereas all ISPs ranked 17 or larger have less than 0.22% of their clients affected. Clearly, something strange is happening in the first 16 ISPs.

Rank	ISP	# of affected	# of total	%
1	Ad-base	613	623	98.39
2	Cincy B.	11,645	12,816	90.86
3	Frontier	30,153	33,452	90.14
4	Hughes	21,838	24,315	89.81
5	Cavalier	3,462	3,912	88.50
6	Spacenet	372	478	77.82
7	FiberNet	409	591	69.20
8	WOW	11,521	16,851	68.37
9	SDN	586	899	65.18
10	Onvoy	547	1,014	53.94
11	MPOWER	259	788	32.87
12	DataPipe	329	1,369	24.03
13	XO	571	2,563	22.28
14	Internap	396	3,442	11.50
15	Level3	1,566	32,806	4.77
16	Parasun	84	4,392	1.91
17	Clearwire	82	37,953	0.22
18	AT&T	113	53,165	0.21
19	Suddenlink	77	38,199	0.20
20	BellSouth	409	214,233	0.19

Table 3: Top 20 ISPs ranked by the percentage of affected clients.

5 What are the Root Causes?

In addition to discovering how many clients are subject to in-flight modifications and where the clients come from, a cloud provider would certainly like to be able to identify the root causes of the modifications. In the previous section, we observed that a high percentage of clients from about a dozen ISPs – such as Frontier, Hughes, and WildOpenWest – are having their web pages modified inflight. This observation alone, however, does not necessarily imply that these ISPs are involved in the modifications. Intead, it could very well be that the clients' local machines are compromised by malicious software – such as *Bahama botnet* [1] – which directs their search queries to the malicious servers.

To identify the root cause, in this section, we conduct detailed correlation analysis. We show that, even though the observations from individual clients are inconclusive, by correlating across the millions of clients, we can reach convincing conclusions.

5.1 LDNS Analysis

Through active probing of the malicious web servers, we discovered that they are very different from general purpose web proxies. These malicious web servers reject most domains other than `example.com` (and a few other cloud companies). This suggests that the affected clients only connect to the malicious web servers when they access `example.com` (and the few other cloud companies). Otherwise, their access to other web sites would be interrupted, which would immediately alert the clients. The only logical explanation is that, when the clients resolve `www.example.com`, the DNS resolution process is compromised in one of the DNS stages. Because the DNS resolution is handled by the clients' local DNS, it is the logical first place to examine for suspicious behavior.

5.1.1 Classifying LDNS

Recall that during the data collection step, we determined each client's LDNS server. Given an LDNS server, we aggregate all the clients that use that LDNS. Based on the percentage of affected clients, we classify each LDNS as either *compromised*, *healthy*, and *inconclusive*.

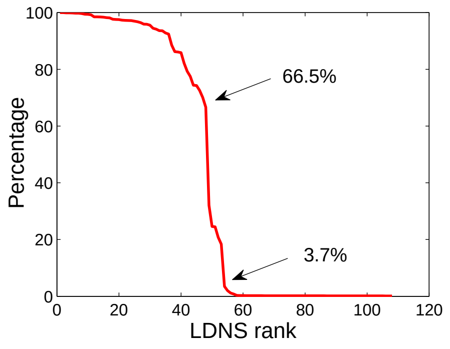

Figure 3: LDNSes ranked by the percentage of affected clients.

Our DNS Echo platform collected 191,479 LDNS IP addresses. Among the LDNS IPs, there are 5,129 associated with the affected clients. We then group these LDNS IPs by /24 prefix and obtain a list of 2,284 prefixes. We remove any prefix that is associated with less than 50 affected clients, which we deem as statistically insignificant. Finally, this gives 108 LDNS prefixes, which belong to 15 ISPs.

Figure 3 plots the rank of the LDNS prefixes against the affected ratio. We see that the affectation ratios dramatically vary among the various LDNSes. We observe there are clear turning-points at 66.5% and 3.7%. Therefore, we conservatively choose two thresholds of 60% and 5% for classification. A LDNS is classified as *compromised* if the affected ratio is lager than 60%, while as *healthy* if the ratio is less than 5%. The LDNSes in-between are classified as *inconclusive*. Figure 4 shows the CDF of the affected ratio. We observe that more than 95% of the LDNSes are either compromised (48) or healthy (55). (Due to space limitation, we do not provide a detailed analysis for the remaining 5 inconclusive LDNSes.)

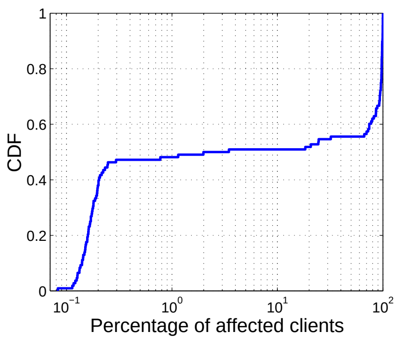

Figure 4: The CDF of the affected clients percentage for LDNS.

5.1.2 Who is behind the compromised LDNSes?

ISP	Official LDNS			Clients		
	cpmzd.	all	%	affect	all	%
Hughes	14	14	100	20,745	21,779	95.5
Frontier	13	14	93	29,899	32,266	92.7
Cavalier	7	7	100	3,362	3,864	87
FiberNet	1	1	100	395	562	70.3
Spacenet	1	1	100	360	368	97.8
Onvoy	3	3	100	496	651	76.1
WOW	3	3	100	11,471	16,732	68.6
Cincy B.	1	1	100	11,644	12,574	92.6
SDN	1	1	100	578	653	88.5

Table 4: ISPs operating compromised (cmpzd) LDNSes.

All LDNSes are *not* deployed by ISPs. For an LDNS deployed by an ISP, we expect it would mostly service clients from the same ISP. Based on this rule, we define an LDNS as *official* to an ISP if more than 50% of the clients using the LDNS are from the ISP. Surprisingly, most – 44 out of 48 – of the compromised LDNS servers are official LDNS servers for 9 ISPs, as shown in Table 4. Furthermore, for each of these ISPs, almost all their LDNS servers are compromised. For instance, Hughes Network deploys 14 official LDNS servers and all 14 are compromised. (The only exception here is one – of fourteen – Frontier LDNS server. Its affected percentage is (10%) and it is classified as inconclusive.)

Thus we can conclude the majority of the inflight modifications are caused by a small number of LDNSes that are responding to DNS queries (for small number of cloud service providers) with malicious servers. The malicious servers then perform the inflight modifications. Moreover, a small number of ISPs (about one dozen) operate these LDNSes; for each such ISP, essentially all of its LDNSes are compromised.

Once all the official LDNS servers of an ISP are identified as comprised, there are certain measures a cloud provider can take to correct the situation. If the LDNSes are compromised because they run the same vulnerable version of DNS software, then the cloud provider can notify the ISP about the problem. Or if the ISP is voluntarily involved in inflight modifications, then appropriate actions, including legal ones, might be taken to stop the malicious activity.

5.1.3 Do the LDNSes discriminate among users?

From our dataset, we observe that a compromised LDNS is sometimes used by clients outside its own ISP. It is of interest to investigate whether the external clients are similarly affected as the clients from within the ISP. Table 5 shows affirmative results. Except for FiberNet and Spacenet, which appear to ban DNS queries from external networks, the compromised LDNS servers in the remaining 7 ISPs also affect external clients.

We further confirm this result by actively probing the compromised LDNS servers. In particular, we send DNS queries to the 44 LDNS servers to resolve `www.example.com` from two vantage points, one at an university and another at an enterprise network. The LDNS servers of 4 ISPs (Fiber, SpaceNet, Cincinnati Bell, and South Dakota) didn't reply. Those of the remaining 5 ISPs all responded with the IP address of malicious servers. Therefore, we conclude that *the compromised LDNS servers don't behave differently depending on the origin of the DNS request; they indiscriminately redirect all clients to the malicious servers*.

ISP	external ISPs	# of clients	# of affected	%
Hughes	6	200	164	82
Frontier	2	1,117	1,094	97.9
Cavalier	2	46	39	84.7
FiberNet	0	0	0	–
Spacenet	0	0	0	–
Onvoy	1	33	23	69.7
WOW	1	22	14	63.6
Cincy B.	2	27	18	66.7
SDN	1	193	146	75.6

Table 5: External Clients from different ISP (external ISP) are Affected.

5.1.4 The Value of Public DNS Servers

So far, we have concluded that for the above 9 ISPs, clients are mostly affected due to compromised LDNS servers. An interesting question is: if a client bypasses the compromised LDNS, will it still be affected? For example, do the ISPs deploy additional transparent network devices that force the clients to connect to the malicious servers? If so, then bypassing LDNS alone would *not* help.

To this end, we now examine whether clients are affected if they use LDNS servers outside of the 9 ISPs. The results are shown in Table 6. It is clear that nearly all the clients using external LDNSes get correct pages. This is very likely another reason, besides improving global traffic management [10], why a cloud provider like Google offers a public DNS service [2]. We believe that *switching to a public DNS service will help circumvent inflight modifications for the clients from the 9 listed ISPs*.

ISP	# of clients	# of affected	%
Hughes	1,140	3	0.2
Frontier	1,093	2	0.1
Cavalier	412	0	0
FiberNet	347	0	0
Spacenet	0	0	0
Onvoy	82	1	1.2
WOW	275	0	0
Cincy B.	387	0	0
SDN	859	5	0.5

Table 6: Clients using external DNS.

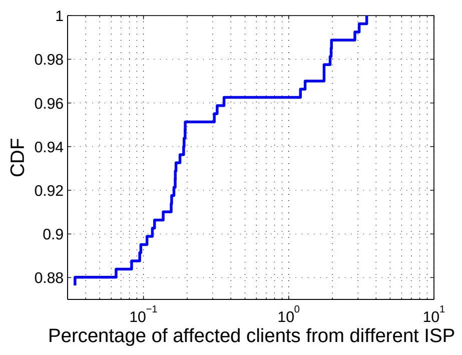

Figure 5: The distribution of affected percentage (by ISP).

5.2 Compromised Host Machines

Now we examine the clients for which their associated LDNSes are classified as healthy. There are 55 healthy LDNSes (i.e., affected percentage is less than 5%). We actively probe these LDNSes by resolving `www.example.com`. Only 5 LDNSes responded, none of which ever responded with an IP address of a malicious server. Therefore, we conjecture that these LDNS servers are genuinely healthy and are *not* the culprits of the inflight modifications. We further conjecture that the affected clients associated with these LDNSes are instead affected by malware in their local hosts; this malware intercepts the DNS query and responds with the malicious server IPs. This, in fact, is how the *Bahama botnet* has been reported to *steal* traffic from popular search engines [1].

To validate our conjecture, we examine all the clients associated with healthy LDNS servers. We group the clients by their ISPs (ISPs with less than 10 clients are removed). Figure 5 plots the CDF of affected percentage of a total 275 ISPs. We observe that only a small percentage of clients are affected within each ISP. For example, more than 95% of the ISPs show affected percentage less than 1.0% and the biggest percentage is 3.7%.

6 Related Work

Reis et al. [14] developed a client-side JavaScript tool, called web tripwire, demonstrated the existence of inflight modifications. Since then, however, there has been lacking of an extensive study about the scale and the root causes of the problem. These are the key questions we explore in this paper.

Inflight modifications can be prevented through end-to-end encryption, such as HTTPS [15]. In HTTPS, HTTP goes over SSL/TLS connection, which offers both integrity and confidentiality. With the recent advance of accelerating HTTPS with commodity hardware [12], switching all traffic to HTTPS appears within reach.

However, HTTPS breaks caching, a key approach used to accelerate Internet today. Stream signing techniques [8] can be used to address the shortcoming, but they cannot handle dynamically generated content – such as Internet search responses – as gracefully as static content.

Instead of detecting modifications, some research efforts focus on identifying similarities of online advertisement, which bear similarity to our Revealer framework. Guha et al. [9] studied the impact of user privacy loss due to online advertising. The focus of these studies, however, is completely different from ours.

Client-side instrumentation techniques are explored extensively in measurement studies. Casado at el. [6] employed JavaScript to collect edge-network properties and study middle-boxes in the Internet. Kiciman at el. [11] proposed a JavaScript tool to monitor web-application performance from end-users' browser. Different from these studies, we develop an extremely lightweight instrumentation technique to overcome deployment barriers and shift focus to analysis, where a tiny bit of information from millions of clients each is aggregated and correlated to reveal the root causes of inflight modifications.

We remark that Netalyzr [13] employs a technique very similar to DNS Echo to discover the local resolvers of arbitrary clients. But different from our study, Netalyzr focuses on aspects closely related to DNS performance, such as extension support, manipulation, proxy and reliability, etc.

Dagon et. al [7] conducted a large-scale scan of IPv4 addresses and discovered over 10 million open recursive (local) DNS resolvers. By probing a subset of these resolvers, they extrapolated that 2.4% are rogue DNS servers, which the authors define as providing incorrect answers to queries for purposes of commercial gain, phishing, or other abuse. Our study differs from that effort, as identifying rogue content servers is much less straightforward, which is why we develop the Revealer platform. More importantly, our correlation analysis identifies the culprits behind the rogue servers with convincing evidence.

7 Conclusion

In this work, we conduct a large scale measurement to examine the inflight modification problem in the wild. We identify candidate rogue servers, and determine the ones performing inflight modifications using our *Revealer* framework. We discover more than 300 malicious servers, which affects more than 65% clients from 9 ISPs. We find that the root cause is not sophisticated transparent in-network services, but instead local DNS servers in the problematic ISPs.

References

[1] Beware the Bahama Botnet. `http://blog.clickforensics.com/?p=314`.

[2] Google Public DNS. `code.google.com/speed/public-dns/`.

[3] Summary of ASes. `http://bgp.potaroo.net/cidr/autnums.html`.

[4] Wireshark. `www.wireshark.org`.

[5] Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology* (2005), UIST '05.

[6] Casado, M., and Freedman, M. J. Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification. In *4th USENIX Symposium on Networked Systems Design & Implementation* (2007), NSDI '07.

[7] Dagon, D., Provos, N., Lee, C., and Lee, W. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of The 15th Annual Network and Distributed System Security Symposium* (2008), NDSS '08.

[8] Gennaro, R., and Rohatgi, P. How to Sign Digital Streams. *Inf. Comput. 165* (February 2001), 100–116.

[9] Guha, S., Cheng, B., and Francis, P. Challenges in measuring online advertising systems. In *Proceedings of the 10th annual conference on Internet measurement* (2010), IMC '10.

[10] Huang, C., Maltz, D. A., Greenberg, A., and Li, J. Public DNS System and Global Traffic Management. In *IEEE INFOCOM* (2011).

[11] Kiciman, E., and Livshits, B. AjaxScope: A Platform for Remotely Monitoring the Client-side Behavior of Web 2.0 Applications. In *21st ACM SIGOPS symposium on Operating systems principles* (2007), SOSP '07.

[12] Kounavis, M., Kang, X., Grewal, K., Eszenyi, M., Gueron, S., and Durham, D. Encrypting the Internet. In *ACM SIGCOMM* (2010).

[13] Kreibich, C., Weaver, N., Nechaev, B., and Paxson, V. Netalyzr: Illuminating the Edge Network. In *Proceedings of the 10th annual conference on Internet measurement* (2010), IMC '10.

[14] Reis, C., Gribble, S. D., Weaver, N. C., and Kohno, T. Automation and Customization of Rendered Web PagesDetecting In-Flight Page Changes with Web Tripwires, 2008.

[15] Rescorla, E. *SSL and TLS: Designing and Building Secure Systems*. Addison Wesley, 2010.

[16] Vratonjic, N., Freudiger, J., and Hubaux, J.-P. Integrity of the Web Content: The Case of Online Advertising. In *Usenix CollSec'10* (2010).