VERISIGN™

OBSERVATIONS ON CHECKSUM ERRORS IN DNS QUERIES TO VERISIGN'S NAME SERVERS

DUANE WESSELS

FEBRUARY 2012

VerisignInc.com

VERISIGN™

CONTENTS

INTRODUCTION: A RECENT PRESENTATION BY ARTEM DINABURG[1] INTRODUCTED THE CONCEPT OF "BITSQUATTING" AS A VARIATION OF DOMAIN NAME TYPO-SQUATTING. THE DINABURG PRESENTATION SUGGESTS THAT BIT-LEVEL ERRORS IN THE HANDLING OF DOMAIN NAMES CAN RESULT IN MISDIRECTED CONTENT. FOR EXAMPLE, A SINGLE FLIPPED BIT CHANGES THE NAME MICROSOFT.COM INTO MICROSMFT.COM. SUCH ERRORS MAY CAUSE A WEB BROWSER TO DISPLAY AN INCORRECT WEB PAGE, OR AN EMAIL TO BE DELIVERED TO THE WRONG SERVER. IN THIS PAPER, WE EXAMINE DNS QUERIES RECEIVED AT AUTHORITATIVE NAME SERVERS RUN BY VERISIGN TO LOOK FOR EVIDENCE OF BIT-LEVEL ERRORS OCCURRING IN THE NETWORK. A SIGNIFICANT ERROR RATE WOULD INDICATE THAT THE "BITSQUATTING" DESCRIBED BY DINABURG REPRESENTS SOME DEGREE OF THREAT TO DOMAIN NAME HOLDERS. WE FIND, HOWEVER, THAT BIT-LEVEL ERRORS IN THE NETWORK ARE RELATIVELY RARE AND OCCUR AT AN EXPECTED RATE.

BIT-LEVEL ERRORS IN DOMAIN NAMES

In an experiment conducted by Dinaburg, domain names such as MICROSMFT.COM and AMAZGN.COM were registered, given an IP address, and an HTTP server installed at that address. Traffic reaching the server was then recorded. Some names, such as LI6E.COM and MIC2OSOFT.COM were selected for being unlikely attributed to typing mistakes.

HTTP requests almost always include a "Host" header, which corresponds to the hostname part of a URL. In Dinaburg's data, 96% of Host headers had the erroneous name. This means that the error happened prior to DNS resolution (i.e., the sending of a query and receipt of the corresponding response). Thus, approximately 4% of the misdirected traffic might be due to errors that occur in DNS resolution.

There are many devices and systems involved in the resolution of DNS names and many places where errors may appear. Consider, for example, a user sitting with a laptop typing in the name of a web site to visit. Possible sources of error include:

- The user's typing
- The user's keyboard (mechanical or electrical errors)
- The application (web browser) processing the user's input
- The stub resolver, when creating the DNS message
- The network (circuits and devices) between stub resolver and iterative resolver (possibly WiFi)
- The iterative resolver application and hardware
- The network (circuits and devices) between iterative resolver and authoritative name servers.
- The authoritative name server application and hardware

1 A. Dinaburg, "Bitsquatting: DNS Hijacking Without Exploitation," Presented at DEFCON 19, August 2011.

Similarly, many DNS queries are generated due to software agents, rather than human users. Examples include loading images on a web page, web crawling, email/spam reputation queries, mailing list delivery, address-to-name lookups, and many more. Sources of error are similar to the list above, except that query names come not from humans and keyboards, but rather from files, caches, databases, and other forms of computer storage.

The cause of errors within most parts of this system can be hard to isolate. Without direct access to all elements, it may be difficult to differentiate between bad typing and faulty memory on an iterative resolver. However, one area where we do have some visibility is the network path between the resolver and our authoritative name servers. Packets sent over the network contain a checksum field, which we can examine to identify errors.

TRANSMISSION ERRORS

User Datagram Protocol (UDP) checksums are an end-to-end feature, designed to protect errors during the transmission of data from a sending host to a receiving host. This may include errors that occur "over-the-wire" (or, perhaps more timely, over-the-fiber) between devices such as switches and routers. It may also include errors that develop inside devices as packets are copied and buffered from one interface to another.

Note that many link layer transmission protocols such as Ethernet, High-Level Data Link Control (HDLC) and Point-to-Point Protocol (PPP) have per-hop checks (usually a Cyclic Redundancy Check (CRC) that are also designed to catch over-the-wire errors [3].

Routers, switches, and other devices do not inspect UDP checksums and will, therefore, happily forward packets with either corrupt data or incorrect checksum values on to the destination.

Although use of checksums with UDP is optional, it is generally enabled by senders. Historically, some found the checksum calculation too burdensome by processors oand it was disabled, at least for some applications (e.g., NFS). These days a number of network interface cards have checksum-offloading features.

UDP CHECKSUMS

The majority of DNS messages are delivered over UDP. With the recent introduction of DNSSEC, some transactions occur over TCP, but this is still a relatively small amount (approximately 1%). TCP has checksums, but this paper focuses only on UDP.

The UDP checksum is a 16-bit field in the UDP header. It is calculated as the one's complement of the one's complement sum of the UDP header, UDP payload, plus a small IP "pseudo-header." It is more formally defined in RFC 768.[3]

To calculate a UDP checksum, the data is cast as a sequence of 16-bit values. These values are then added up. If the sum exceeds 216, the overflow is wrapped and added back in. The final operation is to toggle all bits in the calculated value. A sender sets the checksum field in the UDP header to 0x0000 before calculating. A receiver includes the sender's checksum in its calculation. Since the final step of checksum calculation is to toggle all bits, a correctly received message checksums to 0x0000.

Figure 1 (see next page) shows two very simple UDP checksum calculations. The first example, on the left, shows how a sender calculates a checksum over three 16-bit fields, and how the receiver verifies that checksum. The second example, on the right, demonstrates how the overflow is added back in using one's complement arithmetic.

2 J. Stone and C. Partridge, "When The CRC and TCP Checksum Disagree," ACM SIGCOMM Computer Communication Review, Vol 30 Issue 4, October 2000.

3 J. Postel, "RFC 768: User Datagram Protocol", August 1980.

FIGURE 1

```
Sender      Receiver     Sender      Receiver
              7A9D                     D643
 0377         0377        DD0C         DD0C
 7777         7777        C88E         C88E
+ 0A74      + 0A74      + 8420       + 8420
------      ------      ------       ------
 8562         FFFF       229BA        2FFFD

~ 8562      ~ FFFF        29BA         FFFD
------      ------           2            2
 7A9D         0000      ------       ------

                          29BC         FFFF

                        ~ 29BC       ~ FFFF
                        ------       ------
                          D643         0000
```

FIGURE 2

```
Sender        Receiver      Receiver

                  7A9D          7A9D
 0377             0377          0377
 7777             7773          777F
+ 0A74            0A74          0A74
------          ------        ------
 8562             FFFB         10007
~ 8562          ~ FFFB          0007
------          ------             1
 7A9D             0004        ------
                                0008

                              ~ 0008
                              ------
                                FFF7
```

Figure 2 contains an example showing how a single flipped bit manifests in the checksum calculation. Here, a "7" becomes changed into a "3" in one example, and an "F" in the other. In the first case, the receiver calculates 0004 as the checksum, instead of 0000. Note that the receiver's checksum is equal to the amount that was subtracted by flipping the bit. In the second case, the received checksum is FFF7, which, in one's complement arithmetic, is equal to -0008. Again, this is equal to the difference from changing the 7 to an F.

It is clear that when a single bit is flipped, the checksum indicates the position (mod 16) of that flipped bit. When a single bit is flipped, the receiver calculated checksum is equal to either 2n or 65536-2n. What about the converse? As it turns out, we cannot say with certainty that a receiver checksum equal to 2n was caused by single flipped bit. It is possible for multiple flipped bits to result in a receiver checksum equal to 2n (or 65536-2n). However, it is most likely that such a checksum is due to a single bit flip.

DATA SOURCE

For this study we analyzed 24 hours of DNS query data from five Verisign sites: DFW2, IAD3, LON3, NYC3, and SFO1. The query (and response) packets were captured at the remote sites with port mirroring and were delivered to our collection host over Verisign's global 10G backbone. The data was collected between 00:00 UTC October 8th, 2011 and 00:00 UTC October 9th.

The total number of queries captured is 13,031,158,230.

Since we are looking for errors that occur during transmission, we need to be sure that the backbone itself is not a significant source of errors. In other words, we want to be sure that the queries received at the remote servers are correctly conveyed over our backbone without introducing any additional errors. We do not inspect packets at the site itself (so as not to disrupt site operations), so we cannot measure the amount of backbone errors for received packets. However, we can look for errors in packets sent from our servers, i.e., the DNS responses. Since we trust that our servers always compute correct sender checksums, any differences would have been introduced during transmission over the backbone.

We analyzed **12,957,669,961** DNS response messages[4] from this day and found a single checksum error. The receiver calculated checksum was DFFF, indicating a likely bit flip in position 14. Based on this, we feel that the backbone is a relatively clean channel and not likely to be the source of additional errors.

DATA ANALYSIS

Confident that our port mirroring over the backbone is not a significant source of errors, we turn our attention to analyzing UDP queries. One of the first things we look at is the prevalence of disabled checksums. Recall that the UDP checksum is optional and senders set the field to 0000 when it has been disabled. We found 16,021,232 messages with checksums disabled. This represents 0.13% of all queries received. In other words, **99.87% of queries have checksums enabled.** Next, we look at the number of bad checksums. For each query packet in our data set, we perform the checksum calculation over the data. If it is not equal to 0000, either the data was modified in transit, or the sender placed an incorrect value in the checksum field of the UDP header.

We found 131,412 messages with bad checksums. This represents 0.001008% of all messages. In other words, about **1 in 100,000 DNS queries arrive with a bad checksum.** Such queries are ignored by the server operating system and are not delivered to the name server process. They go unanswered. Figure 3 (see next page) shows the distribution of wrong bits in bad checksums. Normally, the receiver calculated checksum is equal to 0000, so this graph simply shows the number of bits set in the receiver's checksum.

Recall that in the one's complement representation, negating a number is accomplished by flipping all its bits. For example, "+4" is represented as 0004, while "-4" is represented as FFFC. This is why the distribution appears symmetric. A single bit flip either manifests as one or 15 bits set in the receiver's checksum and two flipped bits appear as either two or 14 bits, and so on. Thus, we consider the spike at 15 equivalent to one wrong bit.

The histogram in figure 3 may be modeled as two overlapping binomial distributions.[5] The center-weighted distribution is modeled as B(16, 0.5). That is, it represents the probability of having k out of 16 bits wrong where each bit has a 50% chance of being wrong. When each bit has a 50% chance of being wrong, it is unlikely to have only one out of 16 bits wrong. It is equally unlikely to have 15 bits wrong. It is most likely to have 8 out of 16 bits wrong. For the sake of simplicity, we'll call this the multiple-bits-wrong distribution.

After combining the two spikes at 1 and 15 together, the single-bit-wrong distribution is modeled as B(large, small), which represents the probability that k out of large bits are wrong, where each bit has a small probability of being wrong.

4 The number of responses is about 0.5% less than the number of queries. Some queries are ignored due to being malformed, having source port set to 0, or having an incorrect checksum.

FIGURE 3

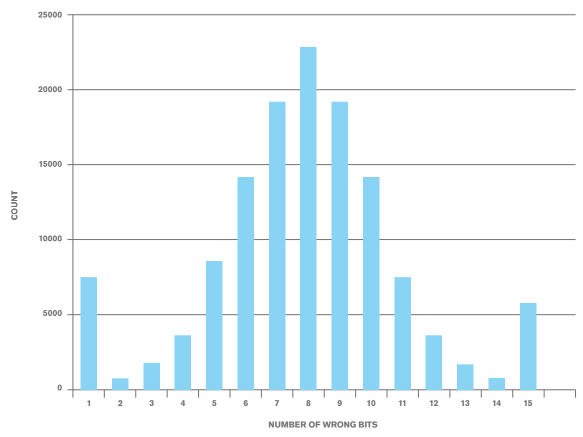
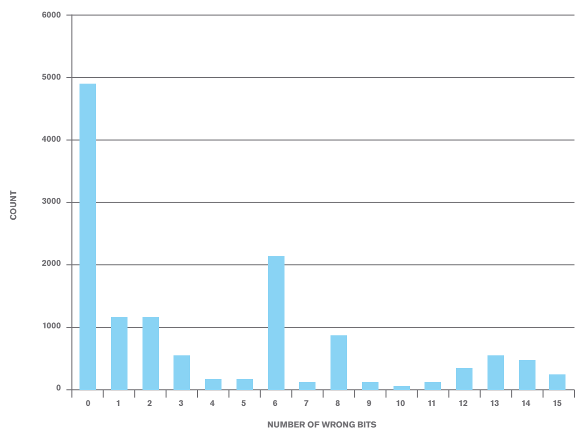


FIGURE 4



In our data set, there are 13,345 occurrences of single-bit-wrong and 118,967 of multiple-bits-wrong errors. These correspond to packet error rates of 1.02×10^{-6} and 9.06×10^{-6} respectively. In other words, about 10% of checksum errors may be attributed to flipping of a single bit while 90% seem to be due to multiple flipped bits.

Given a single-bit-wrong packet error rate of 10^{-6}, and an average DNS query message size of 464 bits, we can estimate the average bit error rate on the network between DNS clients and our servers:

$$10^{-6} = PER$$
$$10^{-6} = 1-(1-BER)N$$
$$BER = 1-\sqrt[N]{1-10^{-6}}$$
$$N = 464$$
$$BER = 2\times10^{-9}$$

A bit error rate in the range of 10^{-9} is about what we'd expect for typical data communication circuits. However, if the data was modified during transmission "over the wire" then it should have been caught by the link-layer CRC or FCS. The fact that we see these errors at the destination host implies that message corruption occurs elsewhere, perhaps inside routers and switches.

Figure 4 above shows the position of the wrong bit among single-bit-wrong checksum errors. The results are counter-intuitive since the distribution is far from uniform. A bit in position #0 is 100 times more likely to be flipped than one in position #10. However, this data does not take into account the characteristics of separate sources or paths through the network. It may be the case that modems from a particular manufacturer contain a bug that causes only certain bits to be flipped. Or perhaps a particular router in use by a large ISP has a faulty memory module and is likely to flip only bits in one position.

5 The binomial distribution tells us the probability of having a certain number of "successes" in a sequence of "yes/no" trials. In this case, the successes are actually "wrong bits" in the checksum.

Figure 5 shows the number of checksum errors per source IP address. Again, intuitively, we might expect errors to be more uniformly distributed than shown here. This is a very heavy-tailed distribution, with a few sources having 1000's of errors and 1000's of sources having just one or two errors.

Figure 6 is another way to look at the amount of queries from each source whose messages have errors. Here we show the fraction of queries from each source having checksum errors. There are some sources (approx. 50), for which 100% of their queries are received with bad checksums. These may be devices with an incorrect implementation of the checksum algorithm. Or they may be devices that attempt to modify DNS queries in flight, but neglect to update the checksum field. A specific example of this behavior is given in the next section.

A surprising number of sources (approx. 400) have checksum errors 50% of the time. As it turns out, we receive duplicated queries from these sources. The first arrives with a bad checksum, followed immediately by a second with a valid checksum. The two messages have identical UDP payloads and differ only in the checksum. An example of this is also given in the following section.

FIGURE 5

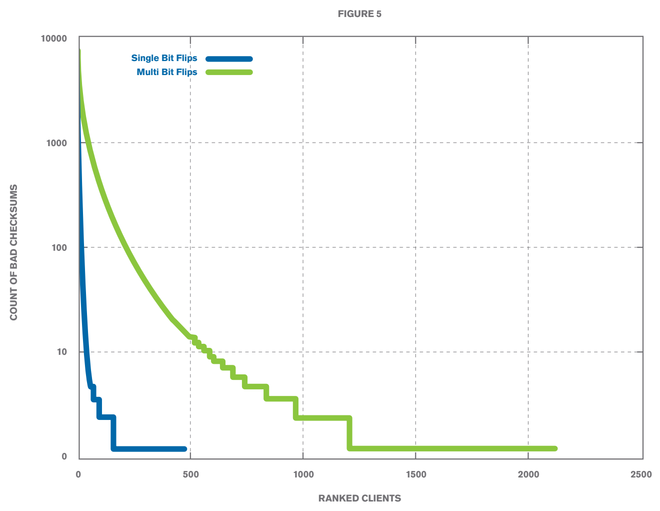
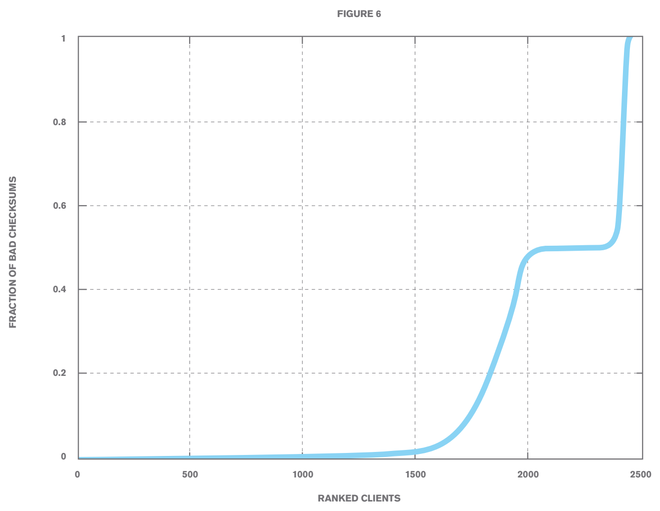


FIGURE 6



EXAMPLES

In this section we present some specific examples of DNS queries with checksum errors.

64.114.x.x

```
00:04:45.977276   IP 64.114.x.x.32768  > 192.58.128.30.53: 3111% AAAA? m.gtld-servers.net. (36)
00:05:09.994614   IP 64.114.x.x.32768  > 192.58.128.30.53: 37477% AAAA? l.gtld-servers.net. (36)
00:05:09.994860   IP 64.114.x.x.32768  > 192.58.128.30.53: 56099% AAAA? c.gtld-servers.net. (36)
00:05:09.994861   IP 64.114.x.x.32768  > 192.58.128.30.53: 43707% AAAA? d.gtld-servers.net. (36)
00:05:09.994862   IP 64.114.x.x.32768  > 192.58.128.30.53: 15202% AAAA? i.gtld-servers.net. (36)
00:05:09.994986   IP 64.114.x.x.32768  > 192.58.128.30.53: 20215% AAAA? e.gtld-servers.net. (36)
00:05:09.994987   IP 64.114.x.x.32768  > 192.58.128.30.53: 32095% AAAA? j.gtld-servers.net. (36)
00:05:11.995590   IP 64.114.x.x.32768  > 192.58.128.30.53: 52463% AAAA? A.ROOT-SERVERS.NET. (36)
00:05:11.995591   IP 64.114.x.x.32768  > 192.58.128.30.53: 56664% AAAA? C.ROOT-SERVERS.NET. (36)
00:05:11.995592   IP 64.114.x.x.32768  > 192.58.128.30.53: 61100% AAAA? D.ROOT-SERVERS.NET. (36)
00:05:11.998389   IP 64.114.x.x.32768  > 192.58.128.30.53: 57738% AAAA? L.ROOT-SERVERS.NET. (36)
00:05:11.998429   IP 64.114.x.x.32768  > 192.58.128.30.53: 63608% AAAA? M.ROOT-SERVERS.NET. (36)
00:05:11.998430   IP 64.114.x.x.32768  > 192.58.128.30.53: 48723% AAAA? E.ROOT-SERVERS.NET. (36)
00:05:11.998542   IP 64.114.x.x.32768  > 192.58.128.30.53: 24474% AAAA? F.ROOT-SERVERS.NET. (36)
```

We received 357 queries with bad checksums from this particular source. We received zero queries with valid checksums. In most cases the query name was for a name ending with gtld-servers.net or root-servers.net. Possibly we only observe queries for these names because this resolver never receives any responses and has not successfully primed its cache. The messages themselves appear to be valid, which implies that the sender calculated checksum is incorrect. In most cases the receiver calculated checksum is off by just one or two bits.

1.9.x.x

```
00:01:16.340277   IP 1.9.x.x.65314        > 192.42.93.30.domain:  21064 A? symantec.com. (30)
00:01:16.340278   IP 1.9.x.x.65314        > 192.42.93.30.domain:  21064 A? symantec.com. (30)
00:01:16.340386   IP 192.42.93.30.domain > 1.9.x.x.65314:         21064- 0/6/2 (233)
00:04:26.451905   IP 1.9.x.x.53372        > 192.26.92.30.domain:  4532 A? www.isupolitiksemasa.net. (42)
00:04:26.451906   IP 1.9.x.x.53372        > 192.26.92.30.domain:  4532 A? www.isupolitiksemasa.net. (42)
00:04:26.452052   IP 192.26.92.30.domain > 1.9.x.x.53372:         4532- 0/2/2 (129)
00:04:26.479518   IP 1.9.x.x.53372        > 192.42.93.30.domain:  4532 A? www.isupolitiksemasa.net. (42)
00:04:26.479521   IP 1.9.x.x.53372        > 192.42.93.30.domain:  4532 A? www.isupolitiksemasa.net. (42)
00:04:26.481201   IP 192.42.93.30.domain > 1.9.x.x.53372:         4532- 0/2/2 (129)
00:04:33.471917   IP 1.9.x.x.51252        > 192.42.93.30.domain:  48411 A? ns.second-ns.com. (34)
00:04:33.471933   IP 1.9.x.x.51252        > 192.42.93.30.domain:  48411 A? ns.second-ns.com. (34)
00:04:33.472040   IP 192.42.93.30.domain > 1.9.x.x.51252:         48411- 0/3/1 (124)
00:05:38.536049   IP 1.9.x.x.49769        > 192.26.92.30.domain:  43629 A? g1.panthercdn.com. (35)
00:05:38.536455   IP 1.9.x.x.49769        > 192.26.92.30.domain:  43629 A? g1.panthercdn.com. (35)
00:05:38.536551   IP 192.26.92.30.domain > 1.9.x.x.49769:         43629- 0/2/2 (103)
00:06:23.624261   IP 1.9.x.x.58598        > 192.26.92.30.domain:  48897 A? mmi.explabs.net. (33)
00:06:23.624363   IP 1.9.x.x.58598        > 192.26.92.30.domain:  48897 A? mmi.explabs.net. (33)
00:06:23.624456   IP 192.26.92.30.domain > 1.9.x.x.58598:         48897- 0/3/3 (157)
00:09:17.086740   IP 1.9.x.x.52698        > 192.42.93.30.domain:  31885 A? www.adobe.com. (31)
00:09:17.086741   IP 1.9.x.x.52698        > 192.42.93.30.domain:  31885 A? www.adobe.com. (31)
00:09:17.086974   IP 192.42.93.30.domain > 1.9.x.x.52698:         31885- 0/4/4 (203)
```

This is an example of a resolver that sends duplicated queries, with one valid and one invalid checksum. The DNS messages are identical. Only the checksum value differs. Closer inspection of the entire packet shows some differences in the IP headers. The packets with a valid UDP checksum all have the IP Identification field set to 0000 and have the DF bit set, whereas, in the invalid message, the DF bit is not set and the Identification field is a random number. Neither of those fields factor into the UDP checksum calculation, however.

There are 400 additional sources with this behavior.

63.146.x.x

```
01:50:39.118685  IP 63.146.x.x.44889 > 192.55.83.30.53: 46274 [1au] MX? latingraf.com. (42)
18:32:06.849303  IP 63.146.x.x.51292 > 192.26.92.30.53: 54241% [1au] A? ns3.covad.com. (42)
01:19:39.607501  IP 63.146.x.x.44309 > 192.42.93.30.53: 51316 [1au] MX? brightroll.com. (43)
01:32:39.348499  IP 63.146.x.x.10057 > 192.42.93.30.53: 44656 [1au] MX? tampatrib.com. (42)
05:48:08.852933  IP 63.146.x.x.13068 > 192.42.93.30.53: 26515 [1au] MX? corwinford.com. (43)
```

This is an example of a source for which we receive a few bad messages, but still more than we should. Out of the 205,439 messages received from this resolver, the five shown above had checksum errors. The DNS queries appear to be normal. The receiver calculated checksum is always off by one bit, either in position 7 or 15.

12.168.x.x

```
00:29:58.244777  IP 12.168.x.x.39965 > 198.41.0.4.domain: 43797+ A? av-begister.bluecoat.com. (42)
00:30:59.237624  IP 12.168.x.x.40036 > 198.41.0.4.domain: 39957+ A? av-Begister.bluecoat.com. (42)
00:31:51.241279  IP 12.168.x.x.40154 > 198.41.0.4.domain: 44309+ A? hb.bluecoat.com. (33)
00:32:51.232745  IP 12.168.x.x.40236 > 198.41.0.4.domain: 36373+ A? smtp.bluecoat.com. (35)
00:57:16.193311  IP 12.168.x.x.42116 > 198.41.0.4.domain: 44821+ A? download.bluecoat.com. (39)
00:58:17.196258  IP 12.168.x.x.42169 > 198.41.0.4.domain: 53269+ A? dowNload.bluecoat.com. (39)
01:22:20.151177  IP 12.168.x.x.43878 > 198.41.0.4.domain: 50453+ A? serFices.bluecoat.com. (39)
04:46:03.804729  IP 12.168.x.x.57706 > 198.41.0.4.domain: 59925+ A? av-Register.bluecoat.com. (42)
08:28:59.426377  IP 12.168.x.x.21790 > 198.41.0.4.domain: 4118+ A? dow^load.bluecoat.com. (39)
08:30:00.423890  IP 12.168.x.x.21842 > 198.41.0.4.domain: 4374+ A? dowNload.bluecoat.com. (39)
08:54:52.380529  IP 12.168.x.x.23358 > 198.41.0.4.domain: 1046+ A? hb.bluecoat.com. (33)
08:55:52.385696  IP 12.168.x.x.23436 > 198.41.0.4.domain: 1302+ A? smtp.bluecoat.com. (35)
09:06:13.362615  IP 12.168.x.x.24262 > 198.41.0.4.domain: 2070+ A? av-begister.bluecoat.com. (42)
09:07:14.359314  IP 12.168.x.x.24344 > 198.41.0.4.domain: 6422+ A? av-Register.bluecoat.com. (42)
09:26:52.327399  IP 12.168.x.x.25746 > 198.41.0.4.domain: 2838+ A? hb.bluecoat.com. (33)
09:27:52.331050  IP 12.168.x.x.25811 > 198.41.0.4.domain: 3094+ A? smtp.bluecoat.com. (35)
```

When looking at the bad queries received from this source, it is relatively easy to spot the errors. Here most of the query names have been changed in ways that are obviously wrong. “begister” instead of “register,” for example. In some queries there is a single upper-case letter in the query name (due to flipping the 0x20 bit). Of the 135 queries from this resolver, 114 had bad checksums.

84.235.x.x/27

```
00:04:18.272383  IP 84.235.x.x.38201  > 192.43.172.30.53: 16852 [16385q] A? ZRZoVynpih.AfaqE2E.cOm.[|domain]
00:05:26.685679  IP 84.235.x.x.55856  > 192.26.92.30.53: 64699 AAAA? pro.hit.GEmlus.pL.aFaqEre.COm. (47)
00:23:16.351460  IP 84.235.x.x.61205  > 192.26.92.30.53: 50267 [16385q] A? AduLTvENUe.nET.[|domain]
00:30:31.525547  IP 84.235.x.x.45802  > 192.26.92.30.53: 47960 Type16385? isvSPPylkq.aFaqE2e.Com. (40)
00:32:20.351339  IP 84.235.x.x.39557  > 192.26.92.30.53: 38826 A (Class 16385)? NkpkrSOneJhEnhxi.CoM. (38)
00:37:51.268343  IP 84.235.x.x.47292  > 192.26.92.30.53: 30906 [16385q] A? wWw.123DOllie.COm.[|domain]
00:42:24.798054  IP 84.235.x.x.48576  > 192.43.172.30.53: 18975 [16385q] A? images.MyfAVoRitegaMes.Com.[|domain]
00:42:24.878279  IP 84.235.x.x.59085  > 192.43.172.30.53: 34573 Type16385? zSHUJMchju.Afaqe2E.COm. (40)
00:57:38.206447  IP 84.235.x.x.64715  > 192.43.172.30.53: 44050 [16385q] A? bBuvfrivvm.AfAqe2E.CoM.[|domain]
01:04:00.591573  IP 84.235.x.x.35768  > 209.112.123.30.53: 31850 [16385q] A? zOY.Gov.[|domain]
01:17:40.462135  IP 84.235.x.x.49549  > 192.43.172.30.53: 65269[|domain]
01:18:26.320153  IP 84.235.x.x.62091  > 192.43.172.30.53: 29304 A (Class 65)? TidSPFtjPerHUuw.nEt. (37)
01:42:16.084257  IP 84.235.x.x.65146  > 192.43.172.30.53: 12095 [16385q] A? MQqIXqdvbc.afaQE2e.cOM.[|domain]
01:42:52.656598  IP 84.235.x.x.59965  > 192.43.172.30.53: 2944 Type16385? uyVaAHllvd.afaqE2e.cOm. (40)
02:05:03.663561  IP 84.235.x.x.30693  > 192.43.172.30.53: 3671 [16385q] MX? RAndoMBiRd.CoM.[|domain]
02:10:20.440502  IP 84.235.x.x.27193  > 192.43.172.30.53: 3782 Type16385? rYFSaFijrl.AfaQE2e.cOM. (40)
```

Numerous sources in this netblock share the same odd behavior. By looking at the query names, it is clear that this resolver has implemented the 0x20 entropy hack,[6] but has taken it too far and appears to be flipping the case bit throughout the DNS message, rather than just over the query name. Some messages state there are 16385 question records. Some are querying for the non-standard type 16385 and some for the obviously bogus classes 65 and 16385.

Most interestingly, it seems that whatever device or software is responsible for flipping the 0x20 bit is doing so after the message was originally sent and it does not update the UDP checksum field.

Numerous

```
 src          | type  | class | name                  | pos
-----------------------------------------------------------------
 200.142.x.x  |  1    | 8193  | tracker.pchome.net    | 13
 200.158.x.x  |  1    | 8193  | www.joshvietti.com    | 13
 200.171.x.x  | 28    | 8193  | k.gtld-servers.net    | 13
 200.234.x.x  |  1    | 8193  | ns3.netboxblue.com    | 13
 187.51.x.x   | 28    | 8193  | dns0.hotchilli.net    | 13
 200.220.x.x  |  1    | 8193  | 2ry:nlqrp.e07j4w0s    | 13
 200.220.x.x  |  1    | 8193  | www.alvodireto.com    | 13
 200.234.x.x  |  1    | 8193  | smtp.doc-tools.net    | 13
 200.100.x.x  |  1    | 8193  | ns.optimum-web.com    | 13
 200.142.x.x  | 28    | 8193  | ns1.navegahost.com    | 13
 200.158.x.x  |  1    | 8193  | www.airdeparis.com    | 13
 200.142.x.x  | 28    | 8193  | auth02.de.folz.net    | 13
 200.142.x.x  |  1    | 8193  | ns2.hospedevix.com    | 13
 200.220.x.x  |  1    | 8193  | www.quiquipost.com    | 13
 200.198.x.x  | 28    | 8193  | ns2.active-dns.com    | 13
 200.158.x.x  |  1    | 8193  | www.wingdamage.com    | 13
 189.110.x.x  |  1    | 8193  | i.root-servers.net    | 13
 189.44.x.x   |  1    | 8193  | dns2.namescout.com    | 13
 200.147.x.x  |  2    | 8193  | luzecorcasting.com    | 13
 187.91.x.x   |  1    | 8193  |  j.root-servers.net   | 13
```

6 http://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00

The Numerous output is from the SQL database used to analyze DNS queries with checksum errors. While paging through the table ordered by query class, a cluster of names of length 18 stands out. There are 316 queries from 210 sources where bit 13 of the query class value has been flipped, changing it from 1 to 8193. There are only 17 additional queries where class=8193 and name length is not 18.

It would seem that some software or device exists out there with a bug that flips bit 13 of the class field when the name length is equal to 18.

CONCLUSION

With a packet error rate on the order of 10^{-5}, the designers of the User Datagram Protocol were wise to include a checksum in the packet header.

Although we cannot possibly know the specific characteristics of all the network paths that bring queries to our servers, we believe that the vast majority of network circuits, whether Ethernet, SONET, Frame Relay, or DS1 are protected by link-layer error detection and/or correction. Furthermore, our results do not have the characteristics that we'd normally attribute to link-layer errors. If the errors occurred during data transmission, we'd expect both the bit-position and errors-per-source distributions to be more uniform than observed.

It seems likely that many observed errors occur either at the source (perhaps due to an incorrect implementation of the checksum algorithm), near the source (due to "middleboxes" that fiddle with message contents), or within the numerous routers and switches on the path between sender and receiver.

We believe that UDP checksums are effective at preventing "bitsquat" attacks and other types of errors that occur after a DNS query leaves a DNS resolver and enters the network. Bitsquat errors that occur prior to entering the network, however, will not benefit from UDP checksums since the sender calculates its checksum over the erroneous data.

VerisignInc.com