
# You can't do that with nslookup: A DNS(SEC) troubleshooting tutorial

Michael Sinatra

Energy Sciences Network

NANOG 53

# What this tutorial is about


- Learning how to use advanced tools to troubleshoot DNS and DNSSEC issues.
  - dig
  - dnscap/nmsg
  - dnsviz
  - others
- Understanding DNS and DNSSEC pitfalls and where you can get into trouble.
- Understanding how DNSSEC complicates the DNS picture.
- Learning how basic troubleshooting techniques can help you solve complex problems.

# What this tutorial is not about


- An introduction to DNS.
- An introduction to DNSSEC. (See Matt Larson's excellent DNSSEC tutorial from NANOG 51 or earlier today.)
- A commentary on the "complexity" of DNSSEC and why we should throw up our arms and run away (we definitely shouldn't).
- A discussion of DNS and/or DNSSEC implementations, and/or of automation tools for DNSSEC.
- I assume a working knowledge of DNS, and also of UNIX, its command line, and how to install and/or compile packages in your favorite modern UNIX variant.

# Why?


- You're here, so you probably have an answer in your head.
- In most organizations DNS is a part-time job.
- Sometimes it barely works.
- Sometimes it doesn't really work but there's nobody to fix it.
- Even if *your* organization has a competent DNS person, your users have to get to other people's resources.
- Ever hear the complaint "I can resolve www.foo.com from {home, $ISP, $other_company} but not here. Our DNS servers give me an error."?
- DNSSEC adds security, and it is important, but it creates more failure modes. Whether or not you deploy DNSSEC, others are doing so, and you still need to understand how it works!

# Let's get started


- Let's look at dig and how it can solve some basic DNS problems.
- Why dig?
  - de facto first-tier DNS troubleshooting tool
  - shows output from servers with relevant flags and other important information
  - part of BIND distribution, and widely available
  - people on mailing lists (bind-users@ dns-operations@) will both use and understand dig output.

# Let's get started


- Why not nslookup?
  - It's old. Really old.
  - It masks important diagnostic information.
  - Frustratingly, a lot of users will initially report a problem with nslookup. You'll quickly see how useless this is.
  - It used to use deprecated DNS protocol extensions to work. (Fortunately that has been excised.)
  - In some circumstances, it gives out *just plain wrong* information.

# nslookup problems


- Why not nslookup?
  - Let's try querying an authoritative-only DNS server with nslookup:

```
michael@michaels-macbook-pro:~$ nslookup
> server adns1.berkeley.edu
Default server: adns1.berkeley.edu
Address: 2607:f140:ffff:fffe::3#53
Default server: adns1.berkeley.edu
Address: 128.32.136.3#53
> cisco.com
Server:     adns1.berkeley.edu
Address:    2607:f140:ffff:fffe::3#53

** server can't find cisco.com: NXDOMAIN
```

# nslookup problems


- Wow, it knows IPv6.
- But it gets the answer wrong!
- NXDOMAIN means that the domain cisco.com doesn't exist *at all!*
- (I am sure that's news to them.)
- Let's see what dig says instead:

# nslookup problems


```
michael@michaels-macbook-pro:~$ dig cisco.com @adns1.berkeley.edu
; <<>> DiG 9.4.3-P3 <<>> cisco.com @adns1.berkeley.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: REFUSED, id: 53060
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cisco.com.                     IN   A

;; Query time: 67 msec
;; SERVER: 2607:f140:ffff:fffe::3#53(2607:f140:ffff:fffe::3)
;; WHEN: Tue Mar  8 21:19:48 2011
;; MSG SIZE  rcvd: 27
```

# dig benefits


```
michael@michaels-macbook-pro:~$ dig cisco.com @adns1.berkeley.edu
; <<>> DiG 9.4.3-P3 <<>> cisco.com @adns1.berkeley.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: REFUSED, id: 53060
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cisco.com.                     IN   A

;; Query time: 67 msec
;; SERVER: 2607:f140:ffff:fffe::3#53(2607:f140:ffff:fffe::3)
;; WHEN: Tue Mar  8 21:19:48 2011
;; MSG SIZE  rcvd: 27
```

# dig benefits


```
michael@michaels-macbook-pro:~$ dig cisco.com @adns1.berkeley.edu
; <<>> DiG 9.4.3-P3 <<>> cisco.com @adns1.berkeley.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: REFUSED, id: 53060
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cisco.com.                     IN   A

;; Query time: 67 msec
;; SERVER: 2607:f140:ffff:fffe::3#53(2607:f140:ffff:fffe::3)
;; WHEN: Tue Mar  8 21:19:48 2011
;; MSG SIZE  rcvd: 27
```

# Understanding dig output: flags


- What are these flags?
- Defined originally in RFC 1035:

```
                                    1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

# Understanding dig output: flags


- What are these flags? Initial RFC 1035 definitions:
  - qr: Query Response (response to query)
  - aa: Authoritative Answer
  - tc: TrunCated (answer is truncated and you need to fall over to TCP)
  - rd: Recursion Desired: In a query, this specifies that the querier wants the server to perform recursion. Its value is copied into the response from the server.
  - ra: Recursion Available: The server is willing to do recursion for this client. Of course, not all servers perform recursion, and those that do, often do not do so for every client. It is generally a best practice to restrict recursion to known clients.
- The presence of the RD and the absence of the RA flag prompts dig to issue a more visible warning: “recursion requested but not available.

# Understanding dig output: status


- The "status" portion of the message is derived from the RCODE. Some common RCODEs are:
  - NOERROR: (0) - no error condition. Does not imply that there is an answer to the query.
  - FORMERR: (1) - format error. The name server did not understand the query.
  - SRVFAIL: (2) - server failure. The server couldn't process the query due to a variety of reasons, which will be discussed in a later slide.
  - NXDOMAIN: (3) - "name error" according to RFC 1035. The *domain* in which the query was made does not exist. *Not to be confused with an empty NOERROR response!*
  - NOTIMPL: (4) - not implemented. The name server basically understands the query, but doesn't implement the query type or some other aspect of the query.
  - REFUSED: (5) - refused. The server refuses to answer the query, for a variety of reasons.
  - These are the RCODEs defined by RFC 1035. There are others having to do with dynamic updates and other additions to the DNS protocol. We may discuss some of these later.
- RCODEs can give significant clues as to why a name server isn't behaving the way you want it to, and dig is one of the few tools that consistently and accurately reports the RCODE.

# A simple dig exercise


- Let's figure out if a name server is answering authoritatively for a zone.
  - There are a few ways to do this, and it gives us a chance to play with some of the many dig options. dig options that affect the query that dig makes or the output that it displays are usually prepended with a "+" instead of the usual UNIX "-". There are some dig options that use the "-" and these generally affect the behavior of the program itself.
  - Let's query some name servers and try to interpret the dig output to see if they are authoritative for a zone. We'll also get a good feel for actual dig output of DNS records.

# A simple dig exercise


- I am going to open a terminal window now and do some dig queries from the command line.
- Whoops, that's a lot of output.  (Wait until we try 'dig any berkeley.edu!)
- There are ways to make it shorter.

# A simple dig exercise


- Wow, that really is *short.*
- It's useful for scripting and piping because it doesn't include any extraneous information or boilerplate, like, say nslookup does. But it also doesn't include all of the useful information that can make dig helpful for troubleshooting.
- Instead, we have can use other flags to have dig omit the additional and/or authority sections.
- Okay, let's look at the +noadditional output.

# A simple dig exercise


```
; <<>> DiG 9.4.3-P3 <<>> +noadditional berkeley.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35444
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 6, ADDITIONAL: 10
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;berkeley.edu.                  IN   A

;; ANSWER SECTION:
berkeley.edu.           600  IN   A    169.229.131.81

;; AUTHORITY SECTION:
berkeley.edu.           172800   IN   NS   adns2.berkeley.edu.
berkeley.edu.           172800   IN   NS   ns.v6.berkeley.edu.
berkeley.edu.           172800   IN   NS   aodns1.berkeley.edu.
berkeley.edu.           172800   IN   NS   phloem.uoregon.edu.
berkeley.edu.           172800   IN   NS   sns-pb.isc.org.
berkeley.edu.           172800   IN   NS   adns1.berkeley.edu.

;; Query time: 16 msec
;; SERVER: 128.32.206.9#53(128.32.206.9)
;; WHEN: Sat Mar 12 18:31:28 2011
;; MSG SIZE  rcvd: 404
```

# A simple dig exercise


```
; <<>> DiG 9.4.3-P3 <<>> +noadditional berkeley.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35444
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 6, ADDITIONAL: 10
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;berkeley.edu.                  IN   A

;; ANSWER SECTION:
berkeley.edu.           600    IN   A    169.229.131.81

;; AUTHORITY SECTION:
berkeley.edu.           172800      IN   NS   adns2.berkeley.edu.
berkeley.edu.           172800      IN   NS   ns.v6.berkeley.edu.
berkeley.edu.           172800      IN   NS   aodns1.berkeley.edu.
berkeley.edu.           172800      IN   NS   phloem.uoregon.edu.
berkeley.edu.           172800      IN   NS   sns-pb.isc.org.
berkeley.edu.           172800      IN   NS   adns1.berkeley.edu.

;; Query time: 16 msec
;; SERVER: 128.32.206.9#53(128.32.206.9)
;; WHEN: Sat Mar 12 18:31:28 2011
;; MSG SIZE  rcvd: 404
```

# A simple dig exercise


- Note that the aa bit is set in this output.
- Note also that if we query the same server over and over again, the TTLs do not decrement.
- [Demonstrate the use of multiple queries to a recursive server to show how ttl decrements and aa bit not set.]
- We could also use +norecurse flag.
- [demonstrate its use]
- Using dig, we can see if a server is answering authoritatively for a domain.
- As you may know, when a server is not answering authoritatively, but it is supposed to be, it is *lame.*

# A simple dig exercise


- Setting query types:
  - Note that omitting a query type leads to the default of type A. (No AAAA by default.)
  - To query for other types, simply include them on the command line.
    - `dig ns berkeley.edu`
    - `dig mx berkeley.edu`
    - `dig aaaa berkeley.edu`
    - `dig txt berkeley.edu`
  - Order does not matter, for the most part:
    - `dig ns berkeley.edu @adns1.berkeley.edu ==`
    - `dig @adns1.berkeley.edu berkeley.edu ns`
  - Try this: `dig any berkeley.edu` on different servers...

# A troubleshooting scenario


- A user reports that they cannot resolve a domain (let's say radiofreebeer.com) from *your* name servers, but they can resolve it from their home or ISP name server.
- If you're on a campus or you *are* a big ISP, you'll probably suspect caching issues.
- But what's causing them? And more importantly, *whose fault is it?*
- Of course, if you're validating DNSSEC (and others are not), it may be a validation failure. We'll talk about that later, but what if that's not the case?
- dig is going to be your best friend.

# A troubleshooting scenario


- First, let's figure out what the authoritative servers say about the zone. You can try this on your laptops if you have dig installed.
- I can never remember the .com name servers, so let's start out querying the root:
- `dig ns radiofreebeer.com @f.root-servers.net`
- Oh, yeah, here's one: d.gtld-servers.net. (Since the root servers are not authoritative for .com, we received a *downward referral.*)
- `dig ns radiofreebeer.com @d.gtld-servers.net`
- `dig a radiofreebeer.com @tinkerdork.org`
- Hmmm. Note the authority section (if there is one). Else:
- `dig ns radiofreebeer.com @tinkerdork.org`

# Scenario: authority mismatch


- Oooh, the NS records are different in the parent zone (com) than in the child zone!
- Moreover, the NS records in the child zone don't work! Try:
- `dig a radiofreebeer.com @adns1.radiofreebeer.com`
- `dig a radiofreebeer.com @adns2.radiofreebeer.com`
- (In real life, the NS names probably won't be such a giveaway, which is why you need dig to tell you what's going on!)

# Sidebar


- What does SERVFAIL really mean?
  - I am not a recursive server and I don't know nuthin about the zone in which ur querying. This may also result in a REFUSED or in an *upward referral* (deprecated).
  - It appears that my administrator is trying to configure the zone, but it is broken, missing, or horribly misconfigured and I just can't give you an answer.
  - It's configured in me correctly, but I am a slave to an upstream master, and I haven't (ever?) been able to do a zone transfer from the master, and/or the zone has expired.
  - In a DNSSEC context: I am configured to do validation and the records I am trying to validate are signed, but they fail validation.
  - Other possibilities?

# Scenario: authority mismatch


- What's happening here?
  - First, note that the consensus interpretation of the DNS standards is that the parent (com) is authoritative for the *existence* of the NS records, but the child (radiofreebeer.com) is authoritative for the *contents* of the NS records.
  - So, most caching name servers will be *child-centric.* They will replace the contents of the NS records that they got from the parent with those they got from the child authoritative server.
  - But since the replacement NSes don't work, the zone has effectively been poisoned--not in a malicious way, but simply in an incompetent way.
  - This has a further implication for the resolution of the A record for radiofreebeer.com. Look at the output again.

# Scenario: authority mismatch


- Here's the problem: The TTL for host[12345].radiofreebeer.com's A record is only 10 seconds, but the TTL for the (bad) NS records is 2 days. So, the A record expires out, but the bad NS records are still in the cache. The caching server tries to look up the A record again but can't because it has bad data.
- This is not a rare problem. It often happens with the admin for a domain changes his/her name servers. The admin updates the registry records but forgets to fix the NS records in the zone itself. When the "old" servers are shut down or repurposed, the problem arises.
- Authority mismatches in general are bad, and this is an example of the most egregious error.

# Scenario: authority mismatch


- Okay, so why did the original report say that *my* name servers were broken and others were working?
  - My servers may be busier and have larger caches. They may have already cached the information whereas a home CPE router may not.
  - Some implementations of caching servers are "parent-centric" and will stick with the parent's NS record contents. This is arguably a standards violation.
  - In the past some servers, upon finding lame delegations, would delete the NS record that was lame and, if all authoritative NS records were deleted, they would refetch the NS records from the parent. This can create big problems with looping queries if there are no NS records in the parent that work! But in this case, they would appear to the client to be working correctly.
  - At any rate, the problem is with the remote domain.
- Note that dig gave us all of the information we needed to solve this problem. Could nslookup have done that?

# A couple short-cuts


- Now that we went through all of those dig commands to look for the authoritative name servers for radiofreebeer.com, let me show you a short-cut:
  - `dig +trace ns radiofreebeer.com`
  - Iterates from the root downward and shows all of the authority records. Here you can easily see the authority mismatch.
- In some cases the problem is not an authority mismatch, but a case where one authoritative server isn't properly getting updated versions of the zone file. In this case, the serial number on the SOA record of the zone would be different on that server. There's an easy way to check this with dig:
  - `dig +nssearch tinkerdork.org`
  - It's sometimes fun to try this on TLDs or ccTLDs as they are in the process of updating.
  - It's also useful to check zones you manage periodically to ensure they are getting updated zones as you expect.

# A different scenario


- A departmental IT administrator reports that some of her users are reporting intermittent problems resolving the vendor management server of a commercial software package her group is using. Occasionally it works, but frequently, the application hangs and reports a "DNS time-out" error. She therefore assumes that it is a DNS problem.
- Of course, you have no idea what the application is trying to do, so where do you start?
  - Sniff the wire!

# A different scenario


- You probably know about the major tools used to sniff an ethernet link, especially tcpdump and wireshark.
  - `tcpdump -v` can give useful DNS information.
  - So can wireshark or `tshark` (command-line version of wireshark).
- You may not know about dnscap and nmsg.
  - dnscap: a pcap-based tool that is exclusively for DNS traffic. It can do regular-expression matches on the DNS message itself.
  - nmsg: an extensible tool that can accept various plugins for different types of traffic (including DNS). One of the plugins can produce an output format similar to dig.

# Output comparison for wire DNS traffic


[show output for tcpdump and tcpdump -v]

# Output comparison for wire DNS traffic


- [show tshark output here]

# Output comparison for wire DNS traffic


- [show dnscap -x -g output here (may also need dnscap -x -6 -g)]

# Output comparison for wire DNS traffic


- [show nmsgtool -i <int>+ -V ISC -T dnsqr output here]
- nmsg is generally used for generating passive DNS information, but it's also used for troubleshooting.
  - You can use nmsg to send pDNS data to ISC and then query their database to see how various domains have changed over time.
  - Captures responses to your nameserver, not client queries.
  - This is something good to get involved in!
  - https://sie.isc.org/

# Back to the scenario


- Note what we saw from the various tools: The application was making queries of type ANY. That can easily cause responses of length greater than 512 bytes.
- Originally, RFC 1035 specified that UDP DNS *messages* must be restricted to 512 bytes. If the message exceeded 512 bytes, implementations must fall over to TCP. Of course, sometimes people block TCP/53 because they think it is only necessary for zone transfers.
- RFC 2671 specified an extension, EDNS0, which allows for much larger responses over UDP.
  - Can lead to UDP fragmentation. But some people block UDP fragments!
- [demonstrate dig +bufsize=4096] Note that some versions of dig may do this by default.

# Back to the scenario


- Note also dig output in a TC situation:

```
~> dig any microsoft.com @2001:470:86b2::3
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.6.3 <<>> any microsoft.com @2001:470:86b2::3
;; global options: +cmd
;; connection timed out; no servers could be reached
```

# Looking at DNSSEC output


- We'll get back to the scenario again in a bit. Let's first look at DNSSEC output with dig. Here are the DNSKEYs for berkeley.edu as reported by dig:

```
berkeley.edu.       300 IN  DNSKEY  257 3 10
    AwEAAbxkCCvANJ8oHUnvYuugq1lbQagoKZb40B1c8mY9eZ3BpGxcoACy
    mCkiIWVnR7ebz8bqfrUXyaFgSL391rX2eQSq75h50llokLnVbu9c/uWH /9dQ
    +5Wofc8O68Vvifdxq841/IkdEc9mKlKSDtgFyVfQWkv/GeZbupK9
    LMuuv71kCtz+P+3tEDwdsZ8Ay00RHG4R0y9K8G+Bac8sMAtL9+3ZHpTl
    ySKj8OZOIW11/TADtomdZ2vdp/WruG5gn1t7omD83QQurmxjniN0aTs8
    imsuvXOst8sYZYMaBZSQeK5kcEltVFcHdRzc1yoA57f8Sjpek7DgvA3U
    0ctXJVOo1Jk=

berkeley.edu.       300 IN  DNSKEY  256 3 10
    BQEAAAAB61Gwv6rGmQ2hNNiM+eVzJe3IfFYemDanzVHpjz4cvBUFk8SS
    ai6fI9MYRh2QbFQHAAQLPSY7ZnGkG+y+nXue4S808f4zk+DX/vujKCap
    BzktbWnyeRP/VHV5N0MZXPD/qvstSS2lmOuRMQYMpTohIzJKvi3dyeZQ
    sBqJR9PNpZc=
```

# Looking at DNSSEC output


- sometimes, it's easier to look at key material in the multiline format. We can use the '+multi' option to dig to get this format:

```
$ dig +multi +noall +answer dnskey berkeley.edu
berkeley.edu.           300 IN DNSKEY 256 3 10 (
                BQEAAAAB61Gwv6rGmQ2hNNiM+eVzJe3IfFYemDanzVHp
                jz4cvBUFk8SSai6fI9MYRh2QbFQHAAQLPSY7ZnGkG+y+
                nXue4S808f4zk+DX/vujKCapBzktbWnyeRP/VHV5N0MZ
                XPD/qvstSS2lmOuRMQYMpTohIzJKvi3dyeZQsBqJR9PN
                pZc=
                ) ; key id = 42697
berkeley.edu.           300 IN DNSKEY 257 3 10 (
                AwEAAbxkCCvANJ8oHUnvYuugq1lbQagoKZb40B1c8mY9
                eZ3BpGxcoACymCkiIWVnR7ebz8bqfrUXyaFgSL391rX2
                eQSq75h50llokLnVbu9c/uWH/9dQ+5Wofc8O68Vvifdx
                q841/IkdEc9mKlKSDtgFyVfQWkv/GeZbupK9LMuuv71k
                Ctz+P+3tEDwdsZ8Ay00RHG4R0y9K8G+Bac8sMAtL9+3Z
                HpTlySKj8OZOIW11/TADtomdZ2vdp/WruG5gn1t7omD8
                3QQurmxjniN0aTs8imsuvXOst8sYZYMaBZSQeK5kcElt
                VFcHdRzc1yoA57f8Sjpek7DgvA3U0ctXJVOo1Jk=
                ) ; key id = 12834
```

# Looking at DNSSEC output


- Notice how the multiline format gives us the key id of the two keys. That's very important when we start looking at signatures.
- Let's look at the other fileds:
  - 256/257
    - 256: The SEP bit is not set on the key. This is typically used as a zone-signing key (ZSK).
    - 257: The SEP bit is set on the key. This is typically used as a key-signing key (KSK).
  - 3: Protocol. 3 == DNSSEC. For DNSSEC DNSKEYs, this must always be set to 3.
  - 10: Algorithm. In this case the algorithm is RSASHA512. (See http://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xml)

# Looking at DNSSEC output


- Look at RRSIGs:

```
adns1.berkeley.edu.    3600    IN  RRSIG   AAAA 10 3 3600
   20110415171839 20110311171839 42697 berkeley.edu.
   BmSJ87lF83V77yrbSrBzs4MZMxEaDd0kEcGbDHxDhvVzESn+JwyT8o3d
   gV9XQdgta9vQfAgTWVb7PUAC8NO+Gi0T1aYZ5QcEBIff0GmHIA8mp5hU
   HuFl3C36wrPboSiW5+7CAMPKqgjfqU06UDr0FCmm+ELyx2SVwcWg1GGo Rys=

adns1.berkeley.edu.    172800  IN  RRSIG   A 10 3 172800
   20110415171839 20110311171839 42697 berkeley.edu. K9HyyI
   +K1DxWd6oY3u+Lh6/p5cliPTvVfkv8VQvHQ5hR7417CuHtLNfO
   FfCZx238IwQyiWbFPAPgEBTeUIwEL8oGNcj4Dx3le4LMkf61MeTJSY0V
   tQBPYLwdiSRFIwu3HPZkxERGGHZ3ih/sbVkN/RkPcI3F957u0AoaOIdk H/g=
```

- You can see the usual TTL, CLASS, and RRTYPE fields. Next is the RRTYPE of the record that the RRSIG is *signing,* then the algorithm, protocol (DNSSEC==3 again), and the TTL *of the record being signed.* Next comes the expiration date of the signature and the date/time that the signature was created, followed by the key id of the key used to sign the record (told you it was important!) and the zone that holds the record.

# Looking at DNSSEC output


- We'll talk more about signature expirations later--it's generally going to be a big part of DNSSEC troubleshooting.
- There's one more record I want to show you: the NSEC record:

```
adns1.berkeley.edu.    300 IN  NSEC    adns2.berkeley.edu. A AAAA
   RRSIG NSEC

adns1.berkeley.edu.    300 IN  RRSIG   NSEC 10 3 300 20110415171839
   20110311171839 42697 berkeley.edu. asjx3nEpKsabaW+K/
   g36ybgDrbZysOG0pdAItQgJRRhy5R5saInUsrue WNoEON5fQhDhvzTv
   +kZXKseX3D7oxOSLVwHI1V1BfLcjUwTNIBmCb2xS
   E6gdy7V7q68dEc6brmFDbjr2CE9QNS9/LdTSC8eOHnjeJbDvPc4j8apb KH8=
```

- Since you already know the DNSSEC basics, you probably understand that the NSEC record provides *proof of non-existence.* What the record above basically says is "there are A AAAA RRSIG and (of course) NSEC records for adns1.berkeley.edu and nothing else until adns2.berkeley.edu, which is the next RR."

# Looking at DNSSEC output


- As you can see, DNSSEC adds a bunch of records and creates a lot of additional output in DNS responses. Try playing around with dig some more to see how large the DNS responses are.
- `dig any <domain>` will give you all of the records for a domain or a hostname, including DNSSEC-related records.
- `dig +dnssec [type] <name>` will give you just the RRSIG for the records you requested. The +dnssec sets the DO (DNSSEC OKAY) bit on the query.
- Note the sizes of these messages, remembering that messages over 512 bytes require either TCP or EDNS0.
- As more domains sign their zones, responses are gonna get bigger!

# Back to the scenario


- What if you see this (tcpdump –s0 –v):

# Back to the scenario


- You find out that your IT administrator has a firewall. Using nmsg, we can see that the TC (truncated) bit is set on the responses. That means that the response could not fit inside a 512-byte message, and tcp failover is required.
  - You can also see the same thing with the +ignore flag on dig.
  - Try `dig +ignore any berkeley.edu`
- You can also see from these responses that there is DNSSEC data in the response. You surmise that the vendor has started signing their zones and that may have "caused" the problem.
- Okay, what question do you ask the IT administrator?

# Back to the scenario


I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

I will not block TCP/53 and I will allow UDP fragments

# Back to the scenario


- But remember that the problems were *intermittent*. Wouldn't that imply that the firewall was not to blame? After all, the FW should be blocking all of the responses.
  - However, caching name servers will sometimes respond to ANY queries entirely from cache. They may not have all of the records in their cache, so they will respond with smaller messages.
  - Some services (e.g. OpenDNS) filter their ANY responses to keep them small. If some of the users in this department configured OpenDNS as one of their resolvers, you might see occasional success.
- But there is another explanation.

# Scenario: Large responses


- You should also ensure that all of the authoritative name servers are able to send you the large response. Test each one of them using dig. If a majority of the authoritative servers for a domain are themselves blocking TCP (or some external firewall is blocking TCP), then you will get a similar problem.
- In general, applications should not do queries for type ANY, but some do. Unfortunately, it's not your job to argue with their developers about what they should and should not do. We do see applications in the wild that attempt to perform these queries.
  - When you sign your zones, you can generally expect to have problems with older versions of qmail attempting to send mail to your users.
- You don't need DNSSEC to get large responses. As of March 2011, 'dig any paypal.com' and 'dig any microsoft.com' yielded responses over 512 bytes.

# DNSSEC overview


- Large responses were our first foray into DNSSEC in this tutorial, and they will generally be the first problem you run into. Since the root and other zones have been signed, the large response issue (people with out-of-date or incorrect firewall rules) has diminished. But there may be more of them lurking out there.
- We have seen a bunch of DNSSEC output of dig. Let's now cover the rest of the bits of DNSSEC we're likely to encounter when troubleshooting.

# DNSSEC overview


- Flags:
  - AD (defined in RFC 2535 “old DNSSEC”): Authenticated Data: The records have been validated.
  - DO: DNSSEC okay: Used in a query to request DNSSEC records pertaining to the query.
  - CD: Checking Disabled: This is set in the query when one doesn’t want a validating name server to attempt to validate the requested records. The bit is also set in the response to indicate that checking has indeed been disabled. This is very useful for troubleshooting as one might imagine.
  - Note that CD and DO can be combined (and often are in a forwarding setup).

# DNSSEC overview


- Validation terms:
  - Valid/Secure: The zone is signed, I have a trust-anchor (or chain of trust) for it, and the signatures validate. I return the answer with the AD bit set.
  - Insecure: Either the zone is not signed or the chain of trust is broken somewhere. I return the answer but don't set the AD bit.
  - Bogus/Validation Failure: The zone is signed, I have a trust-anchor (or chain of trust) for it, but the signatures don't validate. I return a SERVFAIL, unless the CD bit is set on the query, in which case I return the answer and set the CD bit on the answer (and certainly don't set the AD bit).
    - Some implementations can also be configured to ignore validation failures for certain zones and instead treat them as insecure.
  - I will use secure/insecure/bogus to refer to these three situations.

# DNSSEC overview


- Useful dig flags for DNSSEC:
  - +dnssec: Sets the DO bit, which requests DNSSEC records and, if the server validates, requests that the AD bit be set if the zone/ records are secure.
  - +cdflag: Sets the CD bit, which disables validation of a validating name server.
  - Both may be combined. Again, this is useful for troubleshooting.
- Note also that DNSSEC requires EDNS0 support. dig will use EDNS0 (default 4096 byte buffer) even if you don't specify +bufsize=4096.
- Dig will include the following, per RFC 2671:

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
```

# DNSSEC signature scenario


- Let's use dig to troubleshoot a fairly common (unfortunately) DNSSEC problem.
- `dig +dnssec www.expired.tinkerdork.org`
- Use @sns1.burnttofu.net or @sns2.burnttofu.net if you don't have a validating server.
- What happened? SERVFAIL. Unfortunately that could be related to a few things. To rule out (or in) DNSSEC, what query would you use?
- What does the output tell you?

# DNSSEC signature scenario


- The signatures have expired, as you can see from their timestamps.
- This is a common problem, especially where signing is done manually.
- There are also web-based tools that can detect this issue.
  - dnsviz.net
  - dnssec-debugger.verisignlabs.com
  - Both of these are excellent tools for examining DNSSEC problems. Let's run both on our expired.tinkerdork.org domain.
- [spend time demoing the two tools.]

# DNSSEC delegation scenario


- dnsviz and dnssec-debugger are great tools, but they can't always find every problem.
- Here's an example of one where they can't find a problem (or where they couldn't).
- Try `dig +dnssec id.id.tinkerdork.org`
- SERVFAIL, but we don't know why.
- Now try dnssec-debugger first. It looks like it's just an unsigned child zone of tinkerdork.org.
- Let's try dig next:
- `dig ns id.tinkerdork.org @tinkerdork.org`
- `dig id.id.tinkerdork.org @tinkerdork.org`
- Things look okay.

# DNSSEC delegation scenario


- Now let's look for DS records. Maybe the zone is supposed to be signed but isn't:
- `dig ds id.tinkerdork.org`
- Okay, there are no DS records, so it should be a regular unsigned zone. But look closely at the "status" field in dig, which gives the RCODE. It says NXDOMAIN. Recall from an earlier slide what NXDOMAIN means--it means that the entire domain of id.tinkerdork.org doesn't exist!
- Repeat the above dig query with the +dnssec flag. Note the NSEC record. id.tinkerdork.org doesn't exist and DNSSEC proves it!
- So why does the same DNS server give me a response when I query it without DNSSEC? Why do non-validating servers like google properly resolve the domain? (Should we cue the Twilight Zone music?)
- Okay, let's now see what dnsviz.net says for this domain.

# DNSSEC delegation scenario


- Hmm, dnsviz says that there are no delegation records. But the tinkerdork.org nameserver says that there are NS records. (Initially, dnsviz was not able to detect such errors.)
- Remember that delegation records must exist in both the parent and child--and they should be the same, as we have seen.
- Because tinkerdork.org is authoritative for the parent (tinkerdork.org) and the child (id.tinkerdork.org), the delegation issue generally gets swept under the carpet...until DNSSEC comes along.
- Although this is a common mistake, people have been getting away with it for so long, they're not likely to understand that they have been misconfiguring their DNS all along!
- Even without DNSSEC, their party will be over if or when they have another party provide secondary service for parent and not the child. Then the id.tinkerdork.org will intermittently not exist, regardless of the DNSSEC status. DNSSEC does us a favor by forcing the issue.

# DNSSEC NSEC scenario


- You are experiencing intermittent problems resolving another unsigned child domain of a signed parent. Sometimes it resolves and sometimes you get SERVFAILs.
- Such intermittent problems often indicate that some (but maybe not all) authoritative servers are giving out bad information. This is usually a good task for dnsviz, since it checks to make sure that all of the authoritative name servers are responding properly. Let's try it. [demonstrate dnsviz on a domain]
- dnsviz shows us that there a problem with some of the NSEC records in the parent, but it doesn't tell us exactly what. We need dig for that.
- Let's use dig to find the NSEC record that each server is giving out. We may need to set +cdflag to ensure that we can see the records, in case they're failing validation [demonstrate].

# DNSSEC NSEC scenario


- Note that the NSEC records have different case (capitalization) on some of the authoritative servers. Because the case is different, but the signatures are the same, the NSEC records fail validation (bogus) when retrieved from some of the servers.
- Because the NSEC record cannot be validated, it cannot prove the non-existence of the DS record for the child zone, and a validating server will return a SERVFAIL for the child zone.
- This is sometimes caused by a bug in the way that implementations handle case in zones and the shorting (de-FQDN-ing?) of domain names. This bug has been fixed in known implementations.
- In this case, dnsviz pointed us in the direction to look, but we had to use dig to get the full answer.

# Wrapping it up


- No single tool does everything, but if you want to troubleshoot difficult DNS, and especially DNSSEC, problems, you need to move beyond nslookup!
- dig should be your go-to tool, even for everyday use, so that you get a good feel for its options and syntax.
- DNS troubleshooting has always been hard, but it is not insurmountable, even with DNSSEC.
- In fact, DNSSEC forces us to avoid bad behavior that we would otherwise get away with.
- Troubleshooting DNS and DNSSEC will be an even more important skill in the coming months and years.
- ISC's SIE can also be used for troubleshooting!
- Make sure you have the right tools!

# Appendix


- Useful references:
  - http://dnsviz.net
  - http://dnssec-debugger.verisignlabs.com
  - ‘man dig’ from your favorite command line
  - nmsg: ftp://ftp.isc.org/isc/nmsg
  - SIE: https://www.isc.org/community/blog/201011/join-global-passive-dns-pdns-network-today-gain-effective-tools-fight-against-

# Appendix


- dig flags used in this tutorial, or otherwise useful:
  - +norecurse – don't ask nameserver to do recursion
  - +[no]additional, +[no]authority, +[no]answer, +[no]all, etc. – [don't] print the section indicated
  - +short – provide the shortest possible output
  - +trace – trace the authority chain from the root to the specified domain
  - +ignore – ignore the tc bit, if response is longer than 512 bytes and EDNS0 is not being used (default if not using +dnssec)
  - +dnssec – ask server for DNSSEC records and ask server to set the ad bit if DNS data validates (sets the DO bit on the query)
  - +cdflags – set the CD (checking disabled) flag, to tell the server not to try to validate the records, and to return the answer

# Appendix


- dig flags used in this tutorial, or otherwise useful:
  - +multi – use multi-line output; useful for printing DNSKEYs (with IDs)