DNS Deep Dive

```
DIG @8.8.8.8 GROKDNS.COM DS +DNSSEC
```

About the Instructor

- Nathan Isburgh
  - Email is nathan dot isburgh at the domain edgecloud.com ☺
- Unix user/enthusiast 20+ years, teaching it 15+ years
- Pretty much a big nerd :)

About the Course

- 1 day, lectures & labs
  - Lunch hour to be announced
- Breaks about every hour
  - Politely wave at me if I've gone on too long
- Telephone policy
  - Take it outside, please.
- Restrooms/Drinks/Food
  - Facility policy

About the Students

- Name?
- Time served, I mean employed, in the IT field?
  - Speaking of time served… Where was your last vacation?
- Department?
- General Linux skill level?
- Existing DNS knowledge level?
- What are you hoping to take away from this class?

Expectations of Students

- Ask Questions!
- Complete the labs
- Have fun
- Learn something

DNS Purpose, History and Design

The Basic Idea

- DNS - Domain Name System - A system of servers and clients which, among many other functions, map *host names* to *IP addresses* and vice versa
- IP Address - Identification information for nodes on a network
- Host name - human-friendly name for identification of nodes on a network
- Example:

  grokdns.com ⇔ 107.170.20.37

Asking the DNS questions

- How does a client ask the DNS for information?
- Queries!
- A client puts together a *query* to send to a name server. This query formats the request in a consistent manner so the DNS server can process and answer with a response.
- What's in a query? A simple query might be requesting the IP address associated with a domain name, as in the previous example. Another example might ask for the mail exchangers of a domain, to deliver an email message.

A Brief History…

- In the late 60's, ARPAnet was formed by [D]ARPA. This infantile network interconnected a few dozen computer sites around the country, and naming was not a problem because the network was mostly static.
- Then in the 70's, ARPAnet grew to a few hundred hosts, and naming had to be centralized. The Stanford Research Institute (SRI) hosted a file called HOSTS.TXT, which was a flat text database of all machines connected on the ARPAnet.
- Using FTP, administrators periodically downloaded the HOSTS.TXT file to update their local host to address mappings. Can you imagine?

History Continued

- Changes to hostnames and addresses were emailed to the SRI Network Information Center (SRI-NIC).
- This setup was tolerable while ARPAnet was still a small, friendly community of nerds.
- Then TCP/IP came along in the early 80's, and the ARPAnet population exploded.
- All of a sudden, HOSTS.TXT, being maintained and hosted by one group at SRI-NIC was untenable.
  - More changes, more frequently
  - Security, name space collisions, etc

DNS to the rescue!

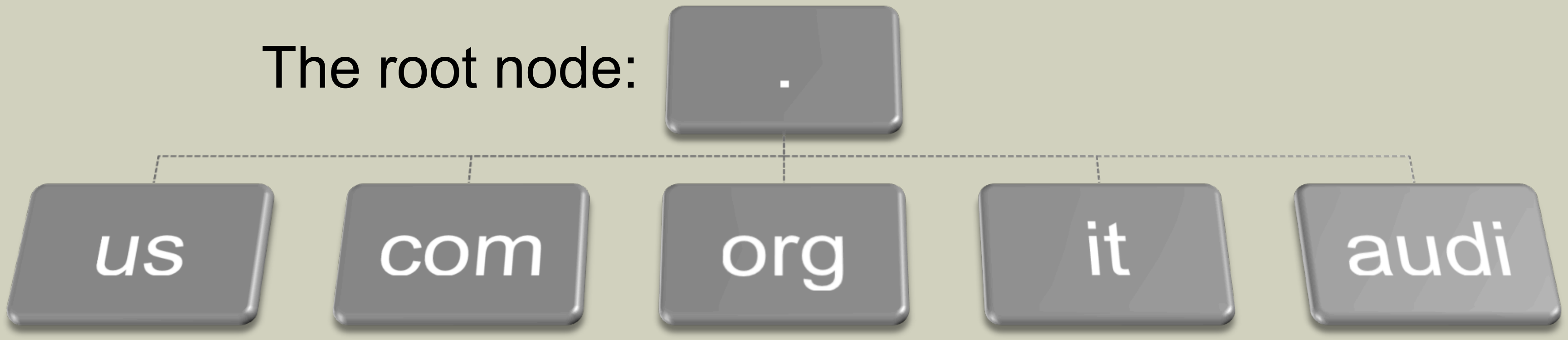
- Paul Mockapetris was charged with designing a replacement for the rapidly failing HOSTS.TXT hack.
- In 1984, Paul released two Requests For Comments (RFC) detailing the design of the Domain Name System.
- This core design still underpins the DNS driving today's internet!
- DNS has since been updated by RFC's 1034 and 1035, as well as many other RFC's covering security and other enhanced features of DNS.

DNS Design Fundamentals

- DNS has a very strong hierarchal structure to it, similar to a filesystem – both are described as an inverted tree.
- At the top is the trunk of the tree, known as the root node, written as a single dot; literally '.'
- Directly beneath the root node are the Top Level Domains (TLDs). The TLDs provide the top-most level of organization of the name space. Think of them as groups or buckets which all domain names fall in to. Some of the TLDs:
  - com, net, org, edu, biz, us, uk, it, jp, info, travel, museum

The Hierarchical Picture

The root node:



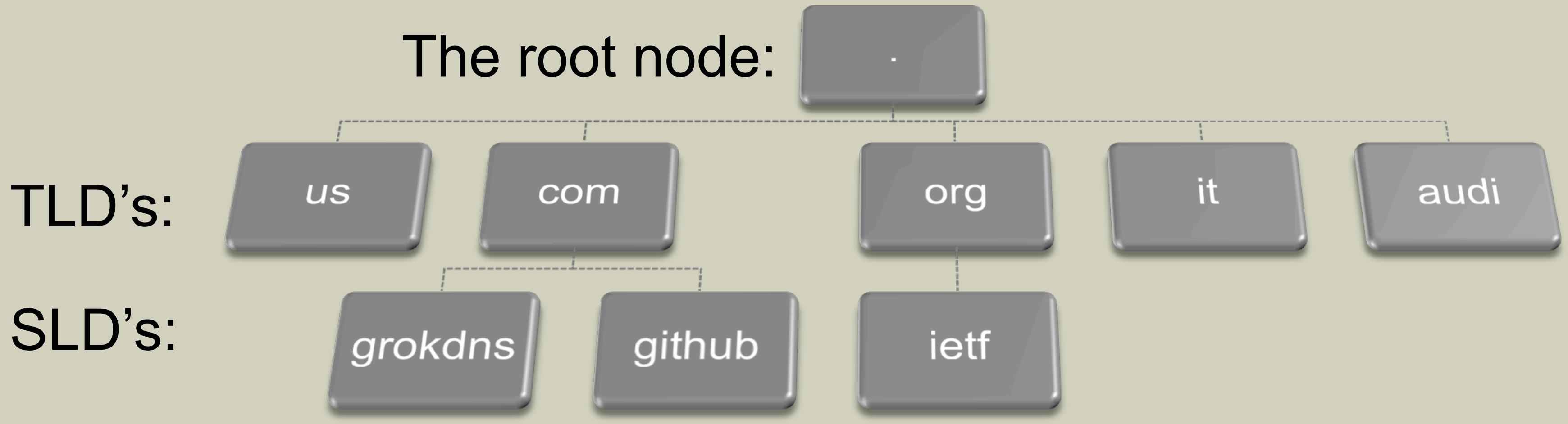
^ Some of the TLD's

- Now imagine the traditional Unix filesystem.. Put / in for the root node, and usr, bin, etc, var, lib, mnt for the TLDs.
- See the similarity?

Second Level Domains

- After the TLD comes the second level domain, only noteworthy in the hierarchy because this is generally what is "owned" when someone registers a domain name: the combination of the second level domain with the top level domain. Examples:
  - grokdns.com
  - ietf.org
  - github.com

The Hierarchical Picture



Inverted Tree Structures

- Due to the underlying shared structure, the similarities between the DNS and a filesystem are numerous:
  - Siblings with the same parent path can not have the same name
    - Only one 'grokdns' under com
    - Can have a 'grokdns' under net, but it would be completely separate
    - This solves the name collision problem!
  - The trunk is called root
  - Trees are recursive structures, meaning each node in the tree is a tree itself, forming a subtree within the overall tree. This is known as a *directory* in filesystems, and a *domain* in DNS.
    - Allows for natural separation points within the namespace:
      - .com for all commercial entities, grokdns.com for the specific 'grokdns' commercial entity, …

Why was it done this way?

- An inverted tree structure:
  - Prevents name collisions! Every node is guaranteed unique!
  - Shares the database naturally across domain owners
  - Easily paves the way for *delegation*
    - The concepts of sub trees allows for simple demarcation of ownership.
    - This is precisely what happens when a domain name is registered: the authority for that name is delegated to the owner, and to join the DNS, the owner will have to operate a name server, contributing their branch to the global tree!
- This is an absolutely critical aspect for the design of DNS! No longer is there one centralized authority for all naming needs (like the old SRI-NIC model).

Naming

- Just like a pathname identifies a location in the filesystem, a domain name identifies a location within the DNS database.
  - In a filesystem, the pathname is specified starting from the root node, separating each directory with a '/', as: `/usr/local/bin`
  - In DNS, the domain is specified in reverse, from the lowest level, back to the root, separating each domain with a '.', as: `www.grokdns.com.`
- The final '.' on a domain name is generally not required, and if it is not present, is assumed to be the root node.
  - Only DNS administrators worry about the trailing dot in configuration files

Delegation

More on Delegation

- DNS administrators can *delegate* authority for a particular domain to a separate entity.
- For example, the grokdns domain was delegated by the com domain to EdgeCloud. Now EdgeCloud manages the entire namespace at and below grokdns.com.
- This is great for everyone - the com DNS admins no longer have to handle change requests for grokdns.com. Further, since the domain was delegated, they don't have to hold the domain database for that branch of the tree. Requests will be passed along to the nameservers EdgeCloud maintains for grokdns.com!

Delegation Everywhere

- Delegation can (and does) occur anywhere in the tree:
  - . delegates com
  - com delegates grokdns
- grokdns has total control over its branch of the tree, so it can create:
  - www.grokdns.com
  - mail.grokdns.com
  - lab09.examples.grokdns.com
  - …
- Notice how the namespace can be used for organization

More Delegation!

- Organization is great. Delegation is better.
  - If examples.grokdns.com starts getting too big for the grokdns.com nameservers (or administrators) to handle themselves, it could be delegated!
  - Then the examples.grokdns.com domain would be completely independent, and could be managed separately from grokdns.com.
- In other words, the *authoritative information* for examples.grokdns.com would be found on the examples.grokdns.com nameservers, not the grokdns.com nameservers.
- The grokdns.com nameservers would only store the *nameserver records* for examples.grokdns.com
  - All queries for examples.grokdns.com to grokdns.com nameservers would get a *referral* to the examples.grokdns.com nameservers

Authority

- Authority decides who is responsible for a given domain
- When a domain is registered, the registrant provides their nameserver hostnames so the registrar can send them to the TLD for delegation purposes
  - When the TLD nameserver receives a request for the subdomain, it will respond with a referral, which includes the nameserver records of the domain, which indicate the DNS servers that are authoritative for the subdomain
  - When the nameservers are within the subdomain (such as ns1.grokdns.com), the parent nameserver will also provide the addresses of the nameservers
    - These special case records are known as *glue records*

Root Nameservers

- The root nameservers are the starting point for the domain name system
- The root nameservers know where all of the TLD nameservers are located
- As you can imagine, the root nameservers are the most important members of the domain name system
- There are currently "13" root nameservers, spread around the globe, on varying networks and operating in multiple geographic locations through anycast routing
  - www.root-servers.org

DNS Queries in Detail

Anatomy of a DNS query

- What really happens when a user types www.grokdns.com in their web browser and presses enter?
  - Browser asks OS for name resolution of www.grokdns.com
  - OS *resolver* consults configuration, possibly returning answer from hosts file, otherwise, assembling DNS query for ISP DNS server
    - The OS will send a *recursive* query, which is how a DNS client asks a DNS server to perform all of the work necessary to get a final answer
  - The ISP DNS server will consult it's cache to speed up responses, but for a complete understanding, we will assume the cache is empty at this time

Anatomy of a DNS query

- The ISP DNS server is preconfigured with the addresses of the root nameservers, which are authoritative for '.'
- Using this information, the ISP DNS server will send a *non-recursive* query to one of the root nameservers, asking it for the address of www.grokdns.com
- The root nameserver is [of course] not authoritative for www.grokdns.com, so it responds with a referral, which does not contain an answer, but does contain authoritative resources to consult next, which in this example would be the nameserver records for com
  - This gets the ISP DNS server one step closer to getting an answer, which is the basic name resolution process of DNS
- Once more, the ISP DNS server will send a non-recursive query to one of the com nameservers asking for the address of www.grokdns.com

Anatomy of a DNS query

- Once again, the com nameserver is not authoritative for www.grokdns.com, so it responds with a referral, providing the nameserver records for grokdns.com
  - Which were provided to the com nameservers by the registrar when the grokdns.com domain was registered
- Now the ISP DNS server has the name server records for com and grokdns.com, both of which will be cached for future queries
- The ISP DNS server will send another non-recursive query, asking the grokdns.com nameservers for the address of www.grokdns.com
- Presuming the www subdomain of grokdns.com is not delegated, this is the end of the referral line:
  - The grokdns.com nameservers will have an answer for the address of www.grokdns.com, which the ISP DNS server will cache and provide to the client, eventually yielding an IP address for the user's web browser to initiate a connection

Query Types

- A <u>recursive</u> query asks a nameserver for the final answer to a question, and relies on the name server to perform the necessary additional queries to provide that information
- A <u>non-recursive</u> or <u>iterative</u> query simply asks a nameserver to provide the best information it has on the query, normally resulting in referral responses
- Recall in the example earlier, the initial query from the OS name resolver (sometimes referred to as a *stub resolver* or *caching resolver*) to the ISP nameserver was recursive. All other requests were iterative. This is the standard method of operation for the DNS.
  - In fact, nameservers are configured to allow or disallow recursive queries depending on the nameserver's purpose

Caching

- Without caching, DNS would be terribly slow, inefficient, and put a ridiculous strain on the root nameservers
- Fortunately, that problem was considered all the way back at the design phase, and so caching is used *everywhere*
  - Watch out, though, it's a double-edged sword!
    - Propagation delays and cache poisoning are serious concerns
- Caching is the technique of storing a local copy of some piece of information in order to have faster access to it
- There are many ways to keep the cache copy "up to date" with the real copy, and in DNS that is achieved by the Time To Live (TTL)

Caching and TTL

- When a nameserver receives a response to a query, it caches that response in order to improve performance on future queries that might need that same information
- Consider, in the previous example, if someone had earlier requested yahoo.com. Then when a request for www.grokdns.com is made, the nameserver would have already cached the nameserver information for the com domain, and would not need to query the root nameservers
- The TTL controls how long cached information is held, anywhere from one second to years
  - The TTL is specified by the authoritative server

Balancing the TTL

- As with everything, setting the TTL is a balancing decision
- A shorter TTL provides for a more rapid propagation of DNS changes, because non-authoritative servers have to expire their cache more often and refresh from the authoritative servers
  - But, it also increases load on the authoritative nameservers due to the increased traffic to refresh caches
- A longer TTL is the inverse - lower load on the authoritative nameservers, but longer propagation times

Reverse Lookups

Reverse Lookups

- Given an IP address, how to find the domain name?
  - Search the entire database one name at a time?
  - Absolutely not! Not feasible, and not even possible!
- in-addr.arpa
  - This is a special domain used to provide IPv4 reverse mapping
- ip6.arpa
  - This special domain provides IPv6 reverse mapping

in-addr.arpa

- Each octet in the dotted quad of an IPv4 address is at a level in the domain tree for in-addr.arpa, descending from root:
  - grokdns.com has an address of 107.170.20.37
  - 37.20.170.107.in-addr.arpa ⇔ grokdns.com

ip6.arpa

- Each nibble of the IPv6 address is at a level in the domain tree for ip6.arpa, descending from root:
  - grokdns.com has an address of 2604:a880:0:1010::1050:9001
  - 1.0.0.9.0.5.0.1.0.0.0.0.0.0.0.0.0.1.0.1.0.0.0.0.0.8.8.a.4.0.6.2.ip6.arpa ⇔ grokdns.com
- Ahhhh, IPv6 addresses…

Why are the Addresses Backwards?

- Notice that the *.arpa names end up reversing the components of the address:
  - 107.170.20.37 ⇔ 37.20.170.107.in-addr.arpa.
  - This is a side effect of how DNS names are written
    - Leaf to root, or most specific to least specific
    - www.grokdns.com goes from www to grokdns to com to root
- IP addresses are the opposite
  - Least specific to most specific, network address to host address
  - 192.168.1.100/24
    - Network address: 192.168.1
    - Host address: 100
- DNS is flexible, so the arpa names could have been reversed, allowing the address and name to line up
  - But what about the way IP addresses are allocated?

Backwards arpa Names

- Recall that IP address space is allocated by network block
- To support delegation, the network address needs to exist "higher" in the DNS hierarchy, i.e. closer to root
- This allows the corresponding arpa domain to be delegated
- Consider a simplified example:
  - Level 3 Communications is allocated the 4.0.0.0/8 network
  - Any address under 4.x.x.x belongs to Level 3, so to create reverse lookup names, 4.in-addr.arpa is delegated to Level 3 by in-addr.arpa
  - Likewise, if Level 3 leases some address space to a customer, say 4.100.100.0/24, then Level 3 will delegate 100.100.4.in-addr.arpa to the customer's nameservers

DNS Tools

Overview

- There are many tools available to query and troubleshoot the DNS
  - Tools to generate queries such as dig and nslookup
  - Tools to query registrar information
  - Tools to test DNS correctness
  - Tools to view and troubleshoot DNS extensions like DNSSEC

dig

- The main workhorse for querying DNS servers is dig
  - dig [@server] [name] [type] [class] [queryopt...]
- Examples
  - dig grokdns.com
  - dig @8.8.8.8 grokdns.com ds +dnssec +qr
  - dig @ns1.grokdns.com grokdns.com soa
  - dig grokdns.com +trace
- The output from dig is a bit ugly because it is designed to be compatible with nameserver database files, known as *zone files*
  - With practice, reading the output will become second nature

nslookup

- Another useful and regularly available tool is nslookup
  - nslookup [name] [server]
- Examples
  - nslookup grokdns.com
  - nslookup grokdns.com 8.8.8.8
- When no arguments are given, nslookup enters a more powerful interactive mode, allowing for custom queries and more
- It is useful to know of nslookup and how to use it because it is common to most operating systems
  - dig is preferred over nslookup for speed and verbosity

Registrars

- Registrars act as the go-between for TLD's
- Registration information is presented through various *whois* information databases
- The whois command can query these databases:
  - whois grokdns.com
  - whois 4.0.0.0
- Notice also that IP address assignments are presented through the whois database, and can be queried directly in most cases, otherwise using special servers
  - whois.arin.net, whois.afrinic.net, whois.apnic.net, …

DNS Correctness Testers

- There are many, many tools available online to perform various tests on DNS infrastructure and configuration
  - dnsviz.net
  - dnsstuff.com
  - mxtoolbox.com
  - ultratools.com
  - dnscheck.pingdom.com
  - viewdns.info

DNSSEC Tools

- There are several DNSSEC troubleshooting tools, of which the most common are:
  - dnsviz.net
  - dnssec-debugger.verisignlabs.com

Lab

<u>Lab/Break time</u>: *20 minutes*

- Explore the various tools discussed in lecture
- Use grokdns.com, or any domain
  - dig grokdns.com
  - whois grokdns.com
  - Run grokdns.com through the testing tools

Resource Records

What is Stored in DNS?

- Resource Records
  - A resource record contains the DNS information about a domain
  - There are many types of resource records, including address records (A), mail exchangers (MX) and name servers (NS).
  - Every domain has at least 2 resource records, an SOA and an NS
    - But that wouldn't be a very useful domain, so there are usually quite a few more records, defining addresses, mail exchangers, canonical names and more.
- The most common types, overviewed next, include:
  - SOA, NS, A/AAAA, CNAME, PTR, MX
- DNSSEC has some additional resource record types:
  - DNSKEY, DS, RRSIG

SOA

- Start of Authority: This resource record identifies authority for a domain
  - Required record at each delegation point to mark new authority
  - Declares an authoritative nameserver along with additional details
  - *+multiline is useful with dig to get more verbose output and comments*

```
$ dig @ns1.grokdns.com grokdns.com soa +multiline

grokdns.com.            60 IN SOA ns1.grokdns.com. dnsadmin.grokdns.com. (
                                2016101204 ; serial
                                600        ; refresh (10 minutes)
                                60         ; retry (1 minute)
                                3600       ; expire (1 hour)
                                60         ; minimum (1 minute)
                                )
```

NS

- Name Server: This resource record identifies authoritative nameservers for a domain
  - Each authoritative nameserver, including the one listed in the SOA, must have an NS record
  - Combined with the SOA, represents the bare minimum resource records required to correctly define a domain with the DNS
  - Returned in the authority section of response, and critical for supporting delegation through referrals

```
$ dig @ns1.grokdns.com grokdns.com ns

grokdns.com.                60 IN NS ns1.grokdns.com.
grokdns.com.                60 IN NS ns2.grokdns.com.
```

A

- Address: This is the workhorse of resource records, as it maps a hostname to an IPv4 address
  - Most DNS tools (dig, nslookup, host) perform A record queries by default

```
$ dig @ns1.grokdns.com grokdns.com a

grokdns.com.                60 IN A 107.170.20.37
```

AAAA

- IPv6 Address: This record maps a hostname to an IPv6 address
  - Commonly referred to as a "quad a" record
  - The four A's in the name are in reference to the size difference between IPv4 and IPv6 address
    - (A) IPv4 has a 32 bit address
    - (AAAA) IPv6 is 4 times larger, with 128 bit addresses

```
$ dig @ns1.grokdns.com grokdns.com aaaa

grokdns.com.            60 IN AAAA 2604:a880:0:1010::1050:9001
```

CNAME

- Canonical Name: This record maps an *alias* hostname to a *canonical* hostname
  - canonical (*adj.*) – accepted as being accurate and authoritative
  - In other words, the canonical name is the "official" name for the host, and the alias is a secondary name
  - In DNS, the canonical name for a host is the name with an A record
  - CNAME records exist to create additional names for the same host
  - In the example below, www.grokdns.com is the alias, and grokdns.com is the canonical name

```
$ dig @ns1.grokdns.com www.grokdns.com cname

www.grokdns.com. 60 IN CNAME grokdns.com.
```

CNAME Records

- Why not just use multiple A records?
  - First, ease of maintenance. If you need 10 names for one machine, defining them with CNAME is easiest if you then need to change the IP address of the machine. Only one change instead of 11.
  - Second, canonicalization. Some services, notably email, will convert all aliases into canonical names. This simplifies mail configuration.
- Note: when an A type query is performed on a CNAME name, the nameserver will respond with both the CNAME record and the associated A record

```
$ dig @ns1.grokdns.com www.grokdns.com a

www.grokdns.com.            60 IN CNAME grokdns.com.
grokdns.com.                60 IN A 107.170.20.37
```

PTR

- Pointer: This record maps a *.arpa hostname to the canonical hostname
  - If an organization has been allocated an appropriately sized network, they can request to have the parent domain delegate the *.arpa name for control and ease of management
  - Otherwise, an organization has to rely on the owner of the containing *.arpa namespace for PTR record management

```
$ dig -x 107.170.20.37

37.20.170.107.in-addr.arpa. 1800 IN PTR  grokdns.com.
```

PTR Record Best Practices

- Remember, there is only one PTR record for a given IP address, and it should always point to the canonical hostname for that host
- Also, as a side note, make sure your *sending* mail servers map both directions exactly. This is important for proper authentication and trust between mail systems:
- If grokdns.com handles outbound email, then:
  - grokdns.com should have an A for 107.170.20.37 ***and***
  - 37.20.170.107.in-addr.arpa. should have a PTR for grokdns.com

MX

- Mail Exchanger: This record declares the hostname for the *receiving* mail server of the domain
  - This resource record value contains two parts
    - The first is a number known as the preference, which prioritizes mailservers
    - The second value is the hostname of the mailserver which accepts email from the internet for the given domain

```
$ dig @ns1.grokdns.com grokdns.com mx

grokdns.com.                60 IN MX 10 grokdns.com.
```

MX Record Preference Values

- MX records allow for enhanced mail routing functionality.
- When an email is shipped out, the server canonicalizes the delivery address. So, for example, bob@www.grokdns.com becomes bob@grokdns.com
- Then the server looks up the MX records for grokdns.com, choosing the record with the lowest preference and attempting delivery to the named mailserver
  - If delivery fails, the next lowest preference mailserver is attempted.
- This allows for backup email servers!

MX Record Preference Values

- But, how does the backup server handle forwarding on the message to the primary?
- Preference values to the rescue again!
- The backup server will compare it's own MX preference value with the list, and discard all records numerically equal to or greater than it's own level, thereby eliminating the chance for mail delivery loops (assuming everything is set up correctly)
- Quite an elegant design, when used

Lab

Lab/Break time: *20 minutes*

- Use dig to explore the various resource records discussed in lecture
- Check A, AAAA, CNAME, SOA, MX and NS records on grokdns.com, ietf.org and/or isc.org

Slaving

Slaving Purpose

- DNS Slave servers are a critical component to the healthy functioning of the DNS
- Slave servers exist to help shoulder the load of requests, and also provide backup resolution services if other nameservers go down
- Slave servers are still *authoritative* for zones
  - The term slave just means that they pull their data from a master server via a zone transfer rather than a local file

How Slaving Works

- The fields in the SOA define slaving configuration
  - The serial number is incremented with each zone change
  - The refresh interval specifies how often a slave checks in with the master for new serial numbers
  - The retry interval specifies how often to start checking if a slave can't connect to the master on one of the refresh checks
  - The expire interval says how long a slave will continue serving a zone's data after losing contact with the master

```
grokdns.com.                60 IN SOA ns1.grokdns.com. dnsadmin.grokdns.com. (
                                    2016101204 ; serial
                                    600        ; refresh (10 minutes)
                                    60         ; retry (1 minute)
                                    3600       ; expire (1 hour)
                                    60         ; minimum (1 minute)
                                    )
```

The Last SOA Field

- The last field in the SOA is alternately marked *minimum* or *negative TTL* depending on the tool
  - Originally, this field was the minimum TTL used for any resource record which did not have a TTL set explicitly (rfc 1035)
  - Now the default TTL is set elsewhere, and this field has been repurposed to the *negative TTL* for the zone
    - Any query for a non-existent name within the zone should cache the *lack* of an answer for the negative TTL specified (rfc 2308)

DNSSEC

Overview

- DNSSEC – DNS SECurity extensions
  - A PKI cryptosystem designed to *authenticate* and maintain *integrity* of DNS data with a series of keys and signatures, all stored directly in the DNS
- Why?
  - Prevent DNS cache poisoning
  - Prevent tampering with valid response data
- Why not? (*none of these are valid reasons today!*)
  - "It's hard"
  - Protocol rewrites in the beginning
  - Higher server load due to cryptographic functions
  - Higher traffic volume due to additional records
  - Higher storage volume due to additional records making for larger zones

DNS Cache Poisoning

- DNS Cache Poisoning is a relatively simple attack that preys on the DNS protocol’s lack of authentication
  - A DNS server will simply believe a DNS response that it receives
- An attacker tricks a DNS server into caching a malicious response through a number of tricks, such as:
  - Flooding the server with queries and spoofed responses
  - Waiting for the server to send a query and then sending a spoofed response before the real server can send it’s response
- Once a malicious record is cached in DNS, an unsuspecting user won’t realize they are talking to the wrong server on the network!

DNSSEC to the Rescue

- DNSSEC provides authenticity to DNS responses by cryptographically signing the zone data, then publishing the signatures and public keys right in DNS
- Key trust is maintained through a rigid hierarchy, each parent zone authenticating the keys for delegated zones, descending all the way from the root servers
  - The root servers sign their own keys and establish the base of trust
  - The root server's key is distributed with the nameserver software, and key changes are communicated via multiple mechanisms such as email, special protocol (rfc 5011) or software updates
- This special root key is known as the *Trust Anchor*

DNSSEC

- Consider grokdns.com
  - Two public/private keypairs were generated on the grokdns.com nameserver
    - One is the Zone Signing Key (ZSK) for signing zone data
    - The other is the Key Signing Key (KSK) for signing the ZSK
  - Both public keys get published in DNS as records of type DNSKEY
  - The private ZSK is used to sign the zone data for grokdns.com
    - Each set of resource records sharing the same name, type and class are known as a Resource Record Set (RRset)
    - Signing is performed on each RRset
      - This is done because a query can not request a partial response – the server will always return all matching records based on the query name, type and class
    - Signatures are published as type RRSIG (Resource Record Signature)

DNSSEC Verification

- The verification process involves:
  - Verifying the RRSIG's with the ZSK provided by a DNSKEY record
  - Verifying the DNSKEY records themselves involves:
    - The ZSK is verified by the KSK
    - The KSK is verified by the parent zone using DS records
- A DS (Delegation Signer) record is found on the parent zone, and stores a hash of the KSK for a child zone
  - The DS record is signed by the parent zone's ZSK
- Like before, the parent zone's ZSK is verified by the parent zone's KSK, which itself is verified by the *grandparent* zone's DS records…
  - The trust model repeats all the way to root and the trust anchor

DNSSEC Algorithms and Key Sizes

- DNSSEC can leverage multiple asymetric algorithms, using key sizes up to 4096 bits
  - RSA, DSA, EC
- Recommended minimum key sizes are 2048 bits for a KSK and 1024 bits for a regularly rolled ZSK
  - Rolling a key means creating a new one and re-populating all of the various signatures and DNSSEC details throughout the DNS
- Signatures are provided through digest algorithms
  - MD5
  - SHA1
  - SHA256
  - SHA512
- The most common and well supported is RSA-SHA1

Proving Non-Existence

- The RRSIG records prove authenticity of existing data, but how do we prove the *non-existence* of a record?
- Two methods:
  - NSEC Records – Next Secure – These records store a list of each actual RRset with a given name, as well as the next name within the domain which contains one or more actual RRsets
    - Major drawback is the ability to "walk" all of the names in a zone by following the NSEC records
  - NSEC3 Records – Next Secure 3 – These records work exactly like NSEC records, except they store the *hash* of the names, preventing zone walking!

Common DNSSEC Queries

- Looking up DS records:

  ```
  dig grokdns.com ds
  ```

  - *Presence of DS records indicates a zone is signed*
- Looking up DNSKEY records:

  ```
  dig grokdns.com dnskey +multiline
  ```

  - *The +multiline formats the output more verbosely with comments describing the key details*
- Verifying DNSKEY records:

  ```
  dig grokdns.com dnskey | dnssec-dsfromkey -f - grokdns.com
  ```

  - *Compare computed DS records with DS records from parent zone*
- Requesting RRSIG/NSEC* records with queries:

  dig grokdns.com +dnssec

  - *The +dnssec tells dig to set the DNSSEC OK bit in the query*

DNSSEC Finale

- Once all of the signatures have been verified end to end, the data is deemed authentic and admitted to cache!
  - No more cache poisoning!
  - No more response tampering!

DNS Troubleshooting

Basics

- The basics of DNS troubleshooting include:
  - A strong knowledge of the DNS design and operation
  - Strong competence with lookup tools like dig or nslookup
- Using these skills, most common problems can be identified readily
  - Improper delegation
  - Missing glue records
  - Propagation delays
  - Invalid DNS data
  - Slaving failures

Lab

- A customer set up lab1.grokdns.com as an A record for 192.168.1.100, but is complaining that they continue seeing the wrong website
- Verify the DNS information and determine if there is a caching issue, an administration issue or other
  - Make sure to identify final cause so you can recommend a fix

Lab

- A customer is complaining that their site is not working
  - The site is: do-not-grokdns.com
  - The authoritative nameserver is ns1.grokdns.com, which appears to be working correctly (verify this)
- Perform a root cause analysis to explain to the customer why they can not get to their site

Lab

- A customer is complaining that their newly migrated website is not showing correctly
  - The site is: edgecloud.com
  - ns1.grokdns.com is happily hosting the domain, but the ip address their browser goes to does not match the IP address ns1.grokdns.com is showing!
- Perform a root cause analysis to explain to the customer why they can not get to their site
- The customer had a second question, stating that before the migration, email sent from their main server at edgecloud.com would occasionally get rejected by some mailservers
  - Please review DNS configuration to see if there might be a mistake