Attacking Distributed Systems
The DNS Case Study

Dan Kaminsky, CISSP

Senior Security Consultant

IP Telephony | Contact Centers | Unified Communication | Services

Your Friendly Investigator

- **Who am I?**
  - **Senior Security Consultant, Avaya Enterprise Security Practice**
  - **Author of “Paketto Keiretsu”, a collection of advanced TCP/IP manipulation tools**
  - **Speaker at Black Hat Briefings**
    - **Black Ops of TCP/IP series**
    - **Gateway Cryptography w/ OpenSSH**
  - **Protocol Geek**

What We're Here To Do Today

- **Discuss vulnerabilities in the design of DNS**
  - **Not going to speak about vulnerabilities of specific implementations**
  - **Will discuss *structural faults* – problems that necessarily had to happen given the semi-anonymous formation of the network**
- **Discuss how these vulnerabilities can inform the design of other systems**

The Subject Of Our Investigation

- DNS: The Domain Name System
  - Created by Paul Mockapetris in 1983
  - Fast, easy, accurate way to, given a host's name, find out it's number ("IP Address")
    - The Internet doesn't run on names any more than the telephone network does – everything is numbered for efficiency.
  - "Internet's equivalent of 411"
    - Actually more critical – people will keep the same phone number for years, while some services change their IP addresses constantly
      - Was becoming a management nightmare to pass around lists of names/numbers
      - "Call 411 by default"
    - Internet has no secure mechanism for sending a fail/redirect message ("the number you've called has been disconnected, the new number is…")

The Nature Of Our Investigation

- **The Question: Is it possible for DNS to do anything more interesting than return numbers from names?**
  - **Simple answer: Of course, it can return mail servers, names from numbers, SPF records, etc.**
  - **Better answer: Why do you ask?**
    - **Second oldest "uncontested protocol" for what it does**
      - **Telnet's moved to SSH, Gopher and FTP moved to HTTP**
      - **Only SMTP is in a similar class**
    - **Globally deployed, universally employed**
    - **Routes everywhere, through pretty much any network.**
    - ***Was heavily queried during recent MS Blaster worm.***
  - **Ultimate answer: Yes, or this would be a very short talk!**
- **The strategy: Does DNS have any unexpected similarities – "*homologies*" – to other protocols I consider interesting?**

Homologies Within The Structure of DNS

- DNS Proxies
  - Those that do not know will ask those that do
  - “Recursive Lookup”
  - Lookups are heavy processes; was necessary to centralize the work
- DNS Caches
  - If one name server does proxy for another, results are not ephemeral, rather they’re cached for a definable amount of time (up to a week in most implementations)
- DNS Routes
  - Those that don’t know, and don’t want to ask those that do, can instead reply with a route recommendation of who else to speak to. Those that receive route recommendations will generally follow them.
  - Used to implement the DNS hierarchy
    - .com routes to doxpara.com routes to www.doxpara.com
  - “Iterative Lookup”

Mapping the Domain Name System

- **Why?**
  - **“Over 150K servers on 64.* alone!”**
  - **Do we have tools?**
  - **dnstracer (DJB)**
    - **[*determine*] *where a given Domain Name Server (DNS) gets its information from, and* [*follow*] *the chain of DNS servers back to the servers which know the data.***
  - **dnstracer (mavetju)**

```
67.15.31.131 (67.15.31.131)
|\___ ns1.speakeasy.net [81.64.in-addr.arpa] (216.254.0.9)
|     |\___ dsl081-064-164.sfo1.dsl.speakeasy.net [164.64.81.64.in-addr.arpa] (64.81.64.164)
```

  - **Heady claims, but these tools only describe *internetwork* relationships, not *intranetwork***

DNS Coalescence

- Of those 150K servers, many are:
  - The same server with multiple interfaces
  - Servers in a “silent hierarchy”
    - Alice maintains her own cache, but if she can’t answer a query, she connects to another upstream server rather than some Internet host. Its cache is checked, and so on.
- Would be very interesting to extract these relationships
  - Dependency checking – which servers trust one another to provide the correct name?
    - Vulnerability scope expansion – which other IP’s, if penetrated, would cause harm to name services?
  - Pretty pictures

Enter The Snoop

- DNS Cache Snooping:
  - Name servers maintain caches
    - Static, for authoritative domains
    - Dynamic, for results from recursively acquired data
  - By disabling the RD(Recursion Desired) bit, clients can search only these caches
    - Not necessary, but “ecological”
  - Possible to make judgements about the environment of a name server by what names it has stored
    - Best paper on the subject: “DNS Cache Snooping” by Luis Grangeia
      - Which mail servers are talking to which, what typos people are going to, where hard-to-find people who only trust a few domains might be, etc.
    - We can inject content into caches, then look for it elsewhere to see if our injection spread

Mapping DNS[0]: Simple, Accurate, Dead Slow

- 1. Query one server recursively with something obscure/unique ("nonced")
  - `dig @64.81.64.164 1234.sitefinder.com`

    …

    `1234.sitefinder.com.  86400  IN  A  64.65.61.123`
- 2. Flood every other server *nonrecursively*, looking for the nonce
  - `dig +norecurse @4.2.2.1 1234.sitefinder.com`

    …

    `1234.sitefinder.com.  IN  A`
- 3. Recoil in horror as you realize this is O(N^2); linking 90K servers to eachother this way requires ~8.1B scans that must be done *before cached entries expire*
  - There must be a better way!

On TTL's

- Cache entries aren't just stored for some amount of time and silently aged out
  - DNS publishes time remaining for cache entries, to make sure the distributed caches all delete old data at roughly the same time
    - The clock visibly runs for cached entries
    - Seconds are quite reasonably standardized ☺
  - If entries have a fixed, constant starting TTL("Time To Live"), then (Starting TTL) – (Measured TTL) = (Time Since Server Queried This Name)
    - 3600 starting - 3580 measured = 20 seconds since query

Mapping DNS[1]: Less Simple, Surprisingly Accurate, FAST

- 1. Acquire large list of servers (demo shortly). Shuffle this list.
- 2. Request same obscure name from all of them. (~100/sec is fine). Note the starting TTL of this name. Record the time each query was sent.
- 3. Analyze responses.
  - Those where TTL(response) == TTL(initial) had no cache to depend on.
  - Those where TTL(response) < TTL(initial) depended on another cache. Use the difference between the two to figure out which caches you were scanning at the time.
    - $initial_ttl = 3600;
    - $sendtime = (int $then) - $start;
    - $recvtime = (int gettimeofday()) - ($initial_ttl - $packet_ttl) - $start;
  - There will be many candidates. So reshuffle and rescan.
    - 2 or 3 should be enough for all but the fastest scanners

Optimizing Time

- Integration of latency measurement
  - Skew between when packet is sent and when reception is noted can degrade detected correlations, especially if jitter is high
  - Solution: Measure latency between sending and receiving
    - Traditional approaches:
      - Send, wait, receive, check how much time elapsed. Slow!
      - Send, store the fact that a packet was sent along with the time it was sent, receive, check difference. Easy in Perl, but inelegant.
    - Scanrand Approach
      - Query packets go out with a DNS ID, that must be reflected back
      - 16 bits of capacity = 65536 potential values = Range for 65 10ms intervals or 6.5 1ms intervals
      - Doesn't alter ability of data to get cached (like putting timestamp in name being looked up)

Combining Approaches

- **Use fast method to show relationships, then slow method to perfect them**
  - **Slow method quite fast at validating theories**
  - **Slow method also much better for differentiating:**
    - **Master/Slave**
      - **When slave has data cached, master has data cached. But when master has data cached, slave may not. This is because master has many slaves.**
    - **Identity**
      - **Whenever one IP has data cached, the other IP has data cached. Either they're doing some strange aggressive cache sharing mechanism or they're the same physical server**
- **DNS caches being remotely visible has other effects…**
  - **Can be used to (anonymously) publish and acquire data, *very very slowly.***

Single-Bit Data Transfer[0]: HOWTO

- **Sending:**
  - **Step 1: Split message into individual bits.**
  - **Step 2: For each byte that will be available for reading, do a recursive lookup against a "start bit" address.**
  - **Step 3: For each bit that is 1, do a recursive lookup against a wildcard-hosted name that identifies that bit.**
- **Receiving:**
  - **Step 1: Do a lookup for the first byte's start bit. If set to 1…**
  - **Step 2: Do non-recursive lookups against names that map against all eight bits. Those names that return answers are 1, those that don't are 0.**
  - **Step 3: Integrate bits into a byte and save. Increment byte counter and return to step 1.**
- **Deleting:**
  - **Optional: Simply do a recursive lookup to clear the 0's**
- **Larger scale transmission is, of course, quite possible**

Tunneling Arbitrary Content in DNS HOWTO [0]

- Note: This isn't anything new
  - Text adventures over DNS
  - Calculators over DNS
  - Bittorrent Seeds over DNS
- Upstream: Encode data in the name being looked up (A,TXT,etc)
  - Restrictions: Total length <253chars, no more than 63 characters per dots, only 63 allowable characters
  - Solution: Use Base32(a-z,0-6) to encode 5 bits per character, ~110 bytes total
  - Example:
    ```
    zjabdbcctvaojbz55mqwe224ceyeltkbhyaasncpljgc53pirtsmuzi
    hcjrw.uujca7ytd3tifmmglrcsl65r3w3ba4mixix6nemd6eulfy2ss
    62xmff3zecv.ttivj2trx642zlrgpbwo2f2glnxk7yxyu3pfeiuvgaw
    c7mijpqn5sh4j.63034-0.id-1187.up.foo.com
    ```
  - Though protocol appears to allow multiple "questions" per packet, no actual implementation parses or forwards such (AA bit problems)

Tunneling Arbitrary Content in DNS HOWTO [1]

- **Downstream: More flexible**
  - ***Standard* DNS packets must be <512 bytes at IP layer**
    - **Prevents IP fragmentation from interfering**
  - **Traditional approach: TXT records**
    - **Restrictions: Minimal. Unstructured, high capacity, provides for subrecords of up to 128 bytes, subrecords are not reordered. Data *probably* needs to be ASCII-compatible.**
    - **Solution: Use Base64(a-z,A-Z,0-9,=,/) to encode 6 bits per character.**
    - **Example:**
      - **"MCaydY5mzxGm2QCqAGLObIAKAAAAAAAACAAAAAECMyaydY5mzxGm2QCqAGLObCwAAAAAAAAAAgAC\010AAIAAgACAAAAAAAAAAAAAACh3KuMR6nPEY7kAMAMIFNlaAAAAAAAAAAlpqVwQ5lVTr9mPCY=\010" "mP5svdFBDwAAAAAAEOVFQJwIxQGfAwAAAAAAAECx5TcAAAAAwL2mNQAAAACIEwAAAAAAAIAAAAz\010BAAAMwQAADtpAQC1A79fLqnPEY7jAMAMIFNlLwcAAAAAAAR0tOruqnPEY7mAMAMIFNlBgA=\010"**

Tunneling Arbitrary Content in DNS HOWTO [2]

- **Naïve approach to tunnel suppression: Lets just censor TXT records**
  - **Data *leaving* network can be much more problematic than data *entering* -- doesn't address that**
  - **Breaks SPF, which encodes itself in TXT**
  - **Don't need TXT for arbitrary content**

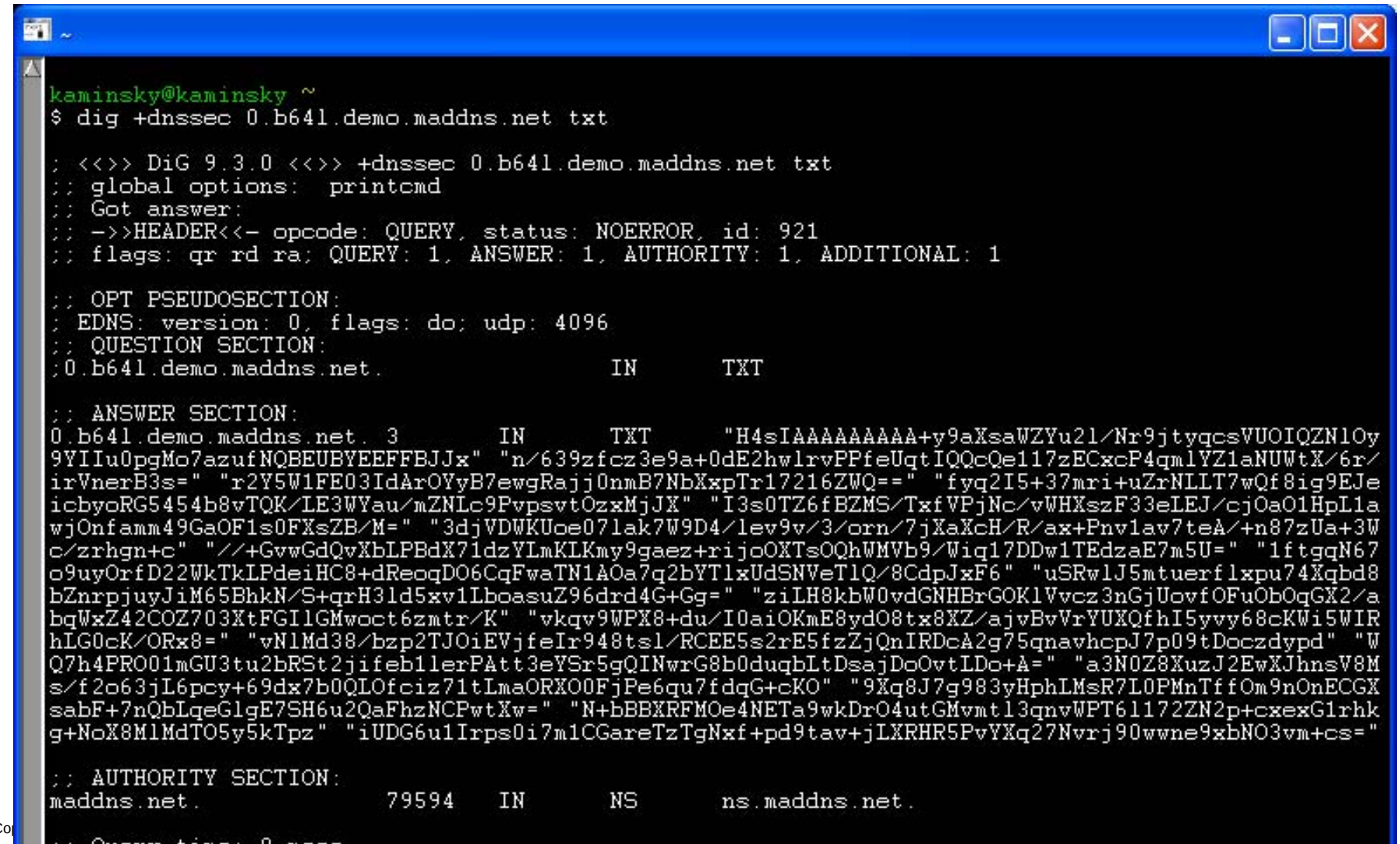
Tunneling Arbitrary Content in DNS HOWTO [2]

- Other downstream approaches
  - MX records (mail)
    - Restrictions: Addresses are shuffled upon delivery to client, to compensate for bad API's (gethostbyname)
    - Solution (care of Dave Hulton): MX records contain *precedence* values, which describe the order in which mail servers should be used. Can also use to describe order in which packets should be reassembled.
  - A records (foo.com -> 1.2.3.4)
    - Restrictions: Addresses shuffled, and no precedence value exists.
    - Solution: We can only fit ~16 IP's into a single response. We can use 4 of 32 bits in each IP to describe the order in which the shuffled addresses should be reassembled. Total capacity becomes ~56 bytes per packet.

Increasing Per-Packet Bandwidth For DNS: EDNS0

- **Size limitations didn't just inconvenience tunnels**
  - **AOL/Yahoo had many, many IP addresses they wanted users to distribute their load across, and they bumped up against the 512 byte limit**
  - **DNSSEC wanted to sign records, but not at the expense of storage capacity**
  - **So capacity had to be increased**
    - **Enter EDNS0, which allows a sender to describe the largest DNS packet his implementation can support.**
    - **Size allowed to exceed IP fragmentation limits (4096 byte advertisements common in wild)**

Effect of EDNS0 on TXT Tunnel Downstream [0]



Effect of EDNS0 on TXT Tunnel Downstream [1]

- **TXT Capacity increases to ~1024 bytes**
  - **While EDNS0 *claims* to allow IP fragmentation in DNS queries, implementations seem to fail if you actually ask them to use it.**
  - **1024 byte transfers = 1453 byte DNS packets. This is too close to the 1500 byte hard limit on Ethernet/IP.**
    - **768 byte encapsulations tend to be reliable.**
      - **More than 3x faster than 220 byte default**
  - **Efficiency improves from 50% to 66%!!!**
    - **Could be worse…there could be an XML schema involved.**

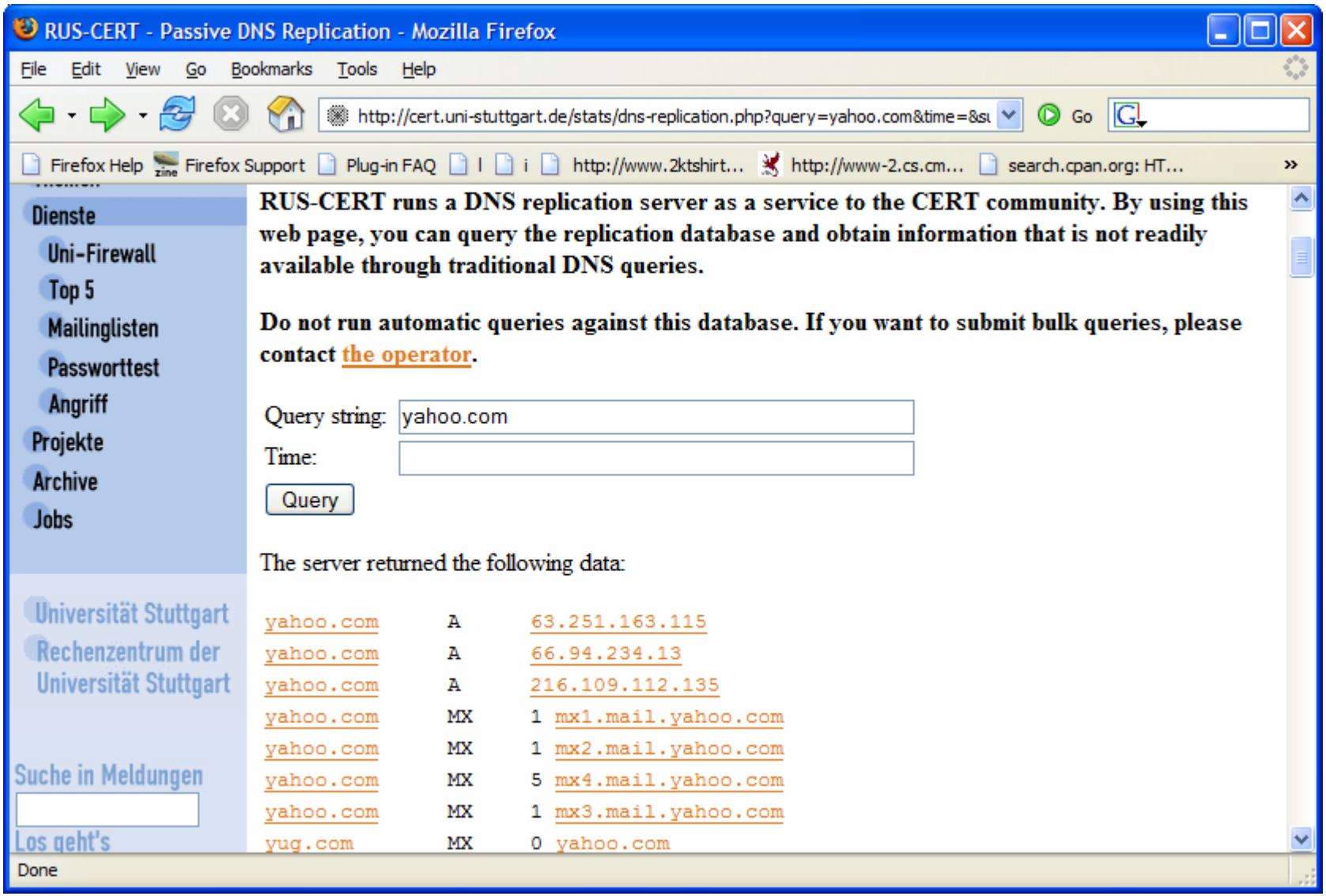
Suppressing DNS Tunnels [0]

- Ongoing research ☺
- Per-packet algorithms will have trouble differentiating odd but legitimate traffic. Think flag, not block
  - Excessively large requests and responses
  - Class D or E IP addresses (224-255.*.*.*)
    - Will break certain multicast implementations!
  - "High Entropy Traffic"
    - DNS names tend to follow English trigraph distributions. Deviations from these trigraphs could be flagged.
      - Interesting things happen with DNS and Unicode. RFC3492 ("Punycode") creates a 1:1 mapping between ASCII names and Unicode that *isn't* Base64. (Yes, there are potential exploits w/ naïve Unicode renderers)

Suppressing DNS Tunnels [1]

- Best (thus far) approaches involves multi-packet analysis
  - Maximum number of queries per minute
  - Maximum number of queries per domain
    - Not enough to limit to different names, as TTL could be set very low and the same name could be flooded
- Tools
  - DNSTop – realtime DNS monitor
  - DNSLogger – “Passive DNS Replication” engine
    - *“Passive DNS replication is a technology which constructs zone replicas without cooperation from zone administrators, based on captured name server responses.”*
    - Should be supporting TXT records soon
    - Used in RUS-CERT project

RUS-CERT DNSLogger Archive



Establishing DNS Tunnels [0]

- There's more to networking than packetized bytes
  - TCP establishes a stream, by which bytes enter one side and exit the other. And either side can talk.
- DNS is not TCP
  - TCP moves bytestreams, DNS moves records
    - Blocks of data
  - TCP lets either side speak first, while in DNS, the server can only talk if the client asks something
  - TCP is 8 bit clean, while DNS can only move a limited set of characters in each direction (Base64 / Base32)
  - This seems so familiar…

Establishing DNS Tunnels[1]

- The semantics of DNS are surprisingly similar to those of HTTP
  - Primary difference – HTTP has unlimited payloads per "download session" but DNS doesn't
    - Exception: Could use AXFR DNS Zone Transfers, which require DNS's TCP mode *but* don't have a maximum size limit
- Many tools have been written with the "lets tunnel everything over HTTP" methodology because it gets through firewalls easier (see first point)
  - SOAP (RPC over HTTP)
  - GNU httptunnel (TCP Stream over HTTP)
- Since DNS has similar semantics, we can pull off similar feats
  - droute: DNS Stream Router

Establishing DNS Tunnels[2]

- Droute: TCP Streaming over DNS
  - "Classic" tool from OzymanDNS
  - Interacts with nomde to allow arbitrary TCP sessions (ordered and reliable bytestreams) to pass over DNS
  - Commonly paired with SSH to allow arbitrary network connectivity using Dynamic Forwarding
- Implementation Details
  - Single threaded state machine. Upstream and downstream unlinked. <1K/s, usable for shell and IM only.
  - Upstream: Wait for data. If any to send, send 110 bytes, wait until remote side acknowledges receipt. Repeat.
  - Downstream: Monitor for data on delay timer. If any to receive, ask for 220 bytes. If so, set delay to minimum and retry. If not, triple delay up until maximum. Repeat.
    - Upstream transmissions do minimize downstream delay

Alternatives to Serialization

- Serial execution makes it slow – only one packet in flight in each direction
  - TCP model is reliable and allows multiple packets in flight – depends on an IP service being available. Could we make DNS look look IP for TCP's use?
    - NSTX (the original net-over-DNS hack) offers this
      - Linux-Only (TUN/TAP based) Kernel Interface for IP<->DNS
      - Unencrypted
    - 5Kb/s: Why?
      - IP fragments are much larger than DNS fragments
      - When dealing with fragmentation, drop any fragment, all fragments must wait
      - This increases latency, which TCP congestion control interprets (bandwidth-delay product) as lowered capacity
  - Need something designed for DNS

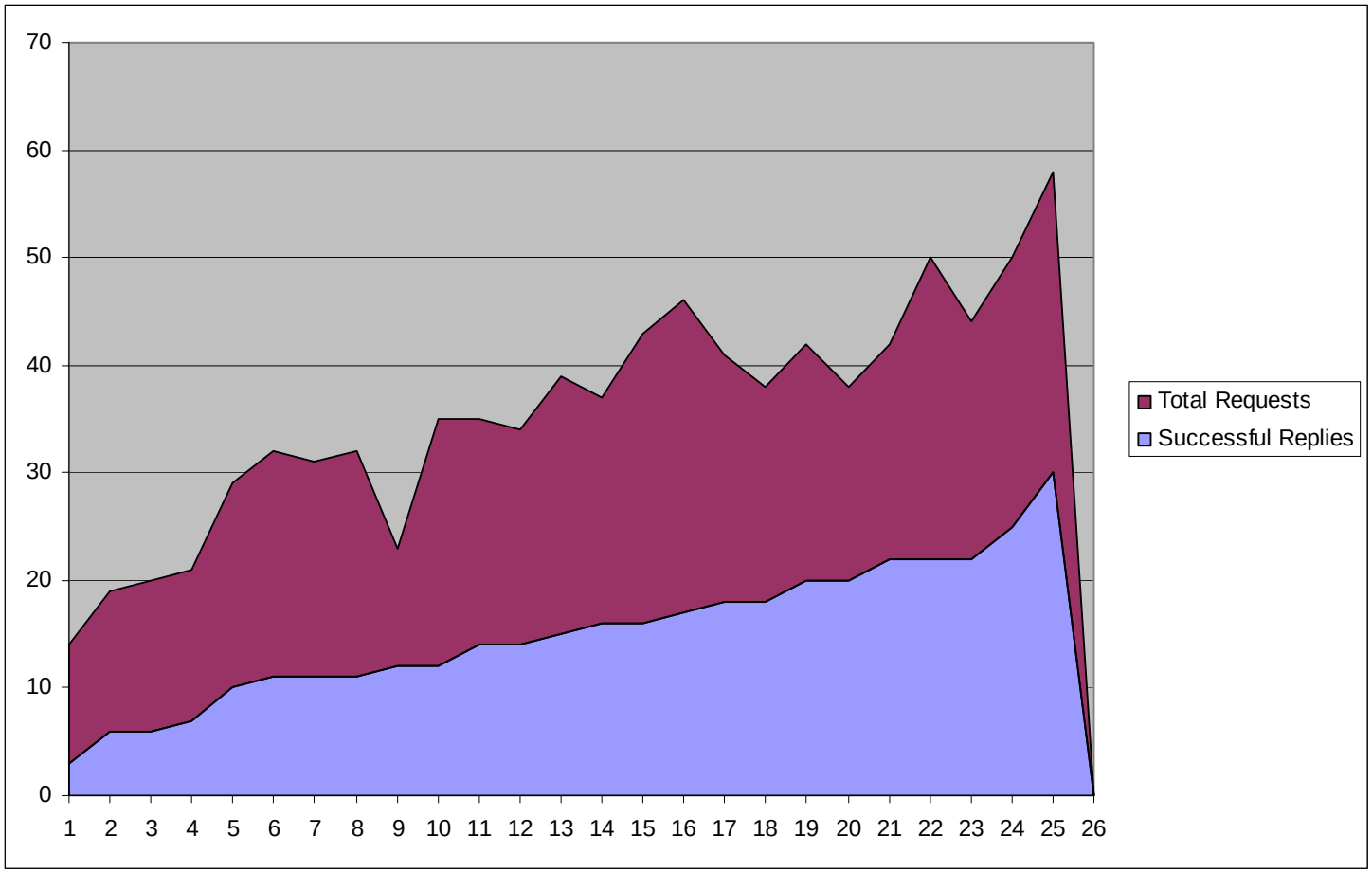
Comparing Operating Layers

- DNS vs. IP as the underlying network (ignore that DNS runs on top of IP)
  - Both IP and DNS transfer records, not bytes
  - Both IP and DNS are unreliable (though DNS clients hide this with a crude retransmit mechanism)
  - IP allows both sides to speak first
    - Firewalls occasionally interfere at the beginning of a L4 session, but once one is established, either side can send a packet
  - IP is 8 bit clean
  - IP networks are built to route large amounts
    - Presumption: Congestion is “strange”
    - That being said, *neither* layer adds significant latency
  - DNS allows me to choose from a large number of routes
    - IP forces me to accept the network’s routes (or those few hops that still support IP Source Route)

Designing an appropriate protocol

- FRP: Fragile Router Protocol
  - FTP -> FSP -> FRP
  - “Fraggle Routing” ☺
- Rate based, not window based
  - Not attempting to discover network capacity – it changes with the degree we stress it.
  - Build to handle DNS’s peculiar size and request model
  - Able to cache large amounts of out-of-order packets and reassemble them as feasible
    - Why out-of-order? Because servers do retransmits, and they don’t appear to be rushable
  - Able to adapt to many routers
  - All complexity lives at the receiver – sender fulfills all requests it can
    - Client/server model forces this

Joys of FRP Design:
The more you ask for, the more you get.



FRP Loop Architecture

- **Send**
  - **If more than $delay_ms milliseconds since the last send, send a query to a random target on the list**
- **Receive:**
  - **While there are packets for us to parse, compare incoming packets to our sentlist and, if we get a response we were looking for, add it to the flush list.**
- **Monitor:**
  - **If more than $stats_interval seconds since the last monitoring (generally, every second), collect statistics.**
  - **If measured success rate is lower than desired, increase interpacket latency. If higher, decrease.**
  - **If we've got any requests that have been out for more than $retrans_delay, put the retransmit at the start of the send queue.**
- **Flush:**
  - **While the flush list contains the required bytes at the left side of our window, flush to the chosen output medium.**

FRP Performance

- CPU bound
  - ~6ms per 768 byte EDNS0
- Performance is still pretty stunning
  - ~22KiB/s streaming for 220 byte packets
  - ~65KiB/s streaming for 768 byte packets
- Surprisingly adaptive
  - Able to adjust to changing network conditions, slow hosts, multiple nameservers
- Demo

Next Steps

- Immature Code
  - Lots and lots of magic constants
  - API still in wild flux
  - OzyResolve get method not embeddable = Not yet ported to SSH over DNS
    - Thank you broken perl threads
  - Eventually need to stabilize and recode in C/C++ for performance
- Protocol Fixes
  - Per-server statistics, better support for multiserver
- New domains
  - Massive Multipath Wireless
  - Heavily Peered Environments
    - Actively requesting individual packets eliminates a number of issues, like dealing with fragmented ranges

Pause for Impact

- So, in summary:
  - We can move arbitrary data.
  - We can bounce it off arbitrary servers.
    - Over 150K on 64.*, over 2M total
  - We can store data in a hop-by-hop basis
  - High speed operations are now feasible.
- "DNS is a globally deployed, routing, caching overlay network running astride the entire public *and private* Internet."
  - You can't ignore it any longer.