MARRAKECH – How It Works: Domain Name Registry Protocols
Monday, March 07, 2016 – 15:15 to 16:45 WET
ICANN55 | Marrakech, Morocco

STEVE: And we've had a good day yesterday and we're having a good day today, so hopefully David will keep it that way.

UNIDENTIFIED FEMALE: No pressure.

STEVE: This is a session on the Domain Registration Protocols and it's presented by David Conrad, ICANN CTO. And with that, I'm going to hand it over to you.

DAVID CONRAD: Thank you, Steve. Yes, I'm David Conrad, ICANN CTO, been working at ICANN this time for about 18 months. This is my second tour with ICANN. Before joining ICANN in August of 2014, I had previously been working at ICANN from 2005 to 2010, where I was the general manager of IANA and eventually was cursed enough to become ICANN's Vice President of IT, which is a job that I'm sure anyone who does IT knows it's not something you'd wish on your worst enemy. But that was my job.

So welcome. This is a tutorial on Domain Name Registration Protocols. I will apologize in advance. I am not the normal person to provide these tutorials, this particular tutorial. That's normally one of my team members, Ed Lewis. Ed was unable to make it to Marrakech, so you get the consolation prize, me. I will try to do him justice, but any mistakes I made are of course my own, not anything that Ed would have put in the slides or anything like that.

Before I start, just out of curiosity – I also have to apologize. I've gotten three hours of sleep in the past two days, so I'm a little wired – how many people here have feel they have a reasonable familiarity with the domain name system, DNS protocols and that sort of thing? Raise your hand. Raise your hand, Julie, come on. Julie Hammer is on ICANN at SSAC and is one of the leading voices on that fine body.

Okay, how many people have any experience with WHOIS and RDAP and EPP and those sorts of protocols? Raise your hand. Okay. That's good to know. I'll probably end up confusing you more than you actually understood before of it. We'll see how it goes.

So, within the DNS, in order to obtain a domain name, you have to deal with the registration ecosystem. A lot of these end-users won't normally interact with. A few, they will. End-users will interact with registrars. They will choose a domain name that's managed by a registry. The registrars sometimes have resellers so that registrants don't talk to the registrars directly. They go through a reseller.

The registries also have a couple of other interactions, something called the Trademark Clearinghouse, TMCH, to ensure that intellectual property is handled correctly. Both registries and registrars have to back up their data, something that's probably a good thing to do especially in the days of CryptoLocker and malware that encrypts your computer.

ICANN mandates something called a data escrow so that if a registry or registrar crashes and burns, there's a way of recovering the data and continuing to provide the domain name service.

I'm going to be talking about all of these. There's another bubble up there that says "Others." There are other protocols, but I'm not going to be talking about them, so don't worry. And there will be a test at the end of this. No, there won't actually.

So, the ecosystem players. There's the registrant, also known as an end-user, frequently known as you or me. The registrant is the holder of the domain name registration. Notice I didn't say owner or anything like that because that gets into really weird questions about ownership of domain names, and I don't want to go there.

There is the registry, and the registry is actually the database of domain names and the mappings of the domain names to the registrants. So, registrant holds a domain name and their contact information, their name, all that sort of stuff is incorporated in that database with a reference to the domain name they hold.

A registrar is the agent that acts between the registrant and the registry. In most cases today, the registrant will not contact the registry directly. They will go through a registrar or a reseller of the registrar. The reason for that is there's a little layer of indirection added that allows for competition at the registrar level. There are multiple registrars that sell services for a particular registry and that allows for competition and lowering price and improving service, and all those wonderful market things.

TMCH is the Trademark Clearinghouse. I mentioned a little earlier it is an institution that was created by ICANN, the first in the world to actually try to manage copyrights across the entire globe. As you're probably aware, every country has their own little laws associated with copyrights. There's never been a repository for trademarks that was global. You can register trademarks across multiple countries and that's how it's usually done, but that didn't really work well for ICANN's needs. So, over time, there was a creation of something called the Trademark Clearinghouse.

And the data escrow agent as I mentioned is a third party back up of registry and/or registrar data. This is a way that, should a registry or registrar have problems, that you as a registrant can be assured that your domain names will be continuing to provide service and you won't lose any of your data.

The others, these are things like privacy and proxy services that can be related to things within the domain name registration ecosystem, but I won't be getting into those.

Hello. Hello. There we go.

UNIDENTIFIED MALE: [Inaudible] point here.

DAVID CONRAD: Point? Oh, not at the screen. Wow. That's amazing how that works. Okay. Did I mention I didn't get much sleep?

So, this graphic, it tends to show sort of the relationships. The registrant as I mentioned is sort of the end-user represented by a [house] as opposed to a person. I guess that's maybe a company, too. And they talk to registrars or the registrant talks to resellers who talk to registrars who talk to TLD registries. And on the other side of the TLD registry, there are the data escrow agent and the Trademark Clearinghouse.

So, now, we'll talk a bit about the protocols that are used in that ecosystem. The one most of you are probably familiar with is obviously the DNS, the Domain Name System protocol itself. It has a little adjunct called DNSSEC to try to make it less vulnerable to the bad guys of the world.

A lot of people probably have used WHOIS in the past to look up information associated with a domain name or an IP address or an autonomous system number.

A replacement for WHOIS called RDAP, data escrow and Trademark Clearinghouse have their own protocols and sort of the non-on-the-wire type protocols.

And there's something called EPP, which stands for the Extensible Provisioning Protocol which unless you are a registry or a registrar, you will not generally use because that's a business-to-business protocol in how registries and registrars talk to each other when they're not screaming at each other.

The Domain Name System, talking about the Domain Name System Protocol … I need to learn to point the right direction. So what is the DNS Protocol? It's a query response protocol that translates a domain name, which is a string of labels that everyone is familiar with, separated by little dots limited to A through Z, zero through nine and the hyphen with some limitations on how those can actually be specified.

How many of you are familiar with internationalized domain names? Can you raise your hand? Have you seen them on the Net? So, it is now possible in the DNS – has been since 2008, I think – to put non-ASCII characters, to put for example Arabic script or Chinese characters or Japanese Hiragana, Katakana, Kanji into the DNS. But the way that's done is it actually translates those scripts or special characters into ASCII and is encoded in something called punycode because that's sort of a pun on Unicode, which is the typographical system where you can actually have all these different scripts and characters. It encodes it into punycode to allow for the use of internalized characters so people who are not so familiar with the ASCII alphabet can use the characters and scripts that they are more familiar with in domain names.

This is actually a really good idea. It's something that has taken a very long time to implement because it turns out it's really, really hard and it's even harder to get the software on the Internet to actually use this stuff. That's one of the challenges that we, within ICANN, are trying to help address at something called Universal Acceptance. There's a Universal Acceptance Steering Group here at ICANN. If anyone is interested in helping to internationalize the DNS, that might be a good place to look into.

Anyhow, getting back to the slide, the DNS translates a domain name into something. That something is some sort of data object that's stored within the DNS. Most commonly, the domain name is translated into an Internet Protocol address, IPv4 or IPv6 address. Those are referenced within the DNS using a type. The type for an IPv4 address is A, stands for Address, and the type for IPv6 is AAAA, which is four A's because an IPv6 address is four times the size of an IPv4 address. The people who came up with this thought it'd be cute.

There are other types that you can look up using the DNS. Mail servers, and the type that you use in the DNS is called MX, which stands for mail exchange. You can actually use the DNS to translate an IP address back into a name. People usually think of translating domain names into IP addresses when they are using the browser to connect to a website. You can actually go the reverse. You can take a IP address, reverse it and append .in-addr.ARPA if it's IPv4 or .IP6.ARPA if it's IPv6 and use a PTR query – PTR stands for Pointer – to look up the reverse.

Fortunately or unfortunately, Internet Service Providers are usually the folks who manage those mappings. They don't generally keep them all up-to-date. You actually have to go in and every time a computer name changes, you have to change the mapping within the DNS. Automation for that is not very common, but it does exist sometimes. In fact, almost every mail server on the planet has to have one of these PTR mappings because if it doesn't, it turns out that a lot of mail companies will think that you are trying to send them spam and just drop the mail on the floor. It turns out that spammers don't usually go through the niceties of configuring PTR records.

You can also ask for where to send more specific questions. So the way the DNS is structured, it's sort of hierarchical. At the very top, you have the root, the next level is top-level domains, below that are second-level domains. I'll talk a bit more about this later, but the NS records are how you get from one level down to the next level.

The query asks for the information and it's sent with what's called a Tuple. There are three parts of the tuple. There is the query name, the name of the thing that you're looking for. The type, whether it's A, quad A, MX, NS, and there are a whole bunch. I think there are 60 types that had been allocated. Only about, I don't know, seven or eight are used commonly.

Then there’s something additional called the Class. When the developers of the DNS were inventing it, they thought it would be good to have different classes that would be sort of separate Internets, and they created the IN class for the Internet. There are two other classes that are defined. One called Hesiod, which was an MIT, Massachusetts Institute of Technology University in Massachusetts in the US. They got a class for their Hesiod information system. Then the third one is also from MIT called Chaos. It was a network protocol that was also developed at MIT, but the only one that’s actually released is the IN class.

Basically, it boils down to a two pool of domain name and type. The response comes back with either the information that’s configured in the DNS server or the information doesn’t exist. Oh, and I should say, if anyone has any questions, feel free to interrupt me at any time. It will probably keep both of us more awake.

Wrong direction. Hello. There we go.

So, the DNS is actually one of the earliest protocols that was invented on the Internet. It was developed around 1983, which purely coincidentally is also about the time I started doing Internet-related stuff. There is no relationship there. However, the age of the protocol does impact the design, its ability to be changed, and in and of itself, it has proven to be somewhat resistant to being replaced.

We've learned a lot in the time since the DNS was actually created about how to develop protocols, things that you want to do to make sure that your protocol is extensible, things that you want to do to make sure that it doesn't sort of block future enhancements and that sort of thing. None of that is in the DNS. It turns out that the DNS itself is actually a really unpleasant protocol to deal with. It's sort of [icky] in a whole bunch of different ways. But it works and it works quite well, and because it works, if it's not broke, don't fix it. They generally do not look for ways to replace it.

Obviously, the domain name registries exist because of the DNS protocol. The DNS protocol itself is a way that you enter and manage the way the data that you have entered is transferred. Learning slowly.

So, as I mentioned, the DNS protocol itself is the way that the registries data is made available. That's obviously critical to the operation of the registry. The important part, though, is the registry database that holds all that information is not the DNS. It's just the database. It has nothing to do with the DNS protocol itself. It's what the DNS protocol uses to fetch the data.

The people who use the DNS, it's essentially impossible to count them. The DNS itself has no mechanism built in that allows you to identify the source of the query. In most cases on the Internet today, actually, I think in about 25% to 30% of the cases, if you're looking on the Internet about where a DNS query is coming from, it'll be coming from Google because about 25% of people on the Internet are actually using Google's public DNS server, 8.8.8.8 or 8.8.4.4.

Because there are so many people who are using the DNS, and to give you sort of an idea of scale, the ICANN runs the L-Root Server. The L-Root Server is distributed across about 180 machines across the planet in, I don't know, how many countries are we in these days? Like 30 countries? Yeah. Something like that.

We get about 25,000 queries per second on the system. There are 13 root servers. So, if you do the math, L gets a little more than others because we are distributed so much but figure in order of magnitude. The root of the DNS is getting about 250,000 queries per second, and that's not the domain that's getting the most queries, perhaps surprisingly. It is a service that has a high load, a large volume of users. Pretty much everyone on the Internet uses the DNS one way or another, and the senders of the queries are essentially anonymous.

Question? No? Yes?

UNIDENTIFIED MALE: Yesterday, in the—

UNIDENTIFIED MALE: I'm sorry just pause one second. We have people on the remote and you have the only mobile mic.

DAVID CONRAD: Oh, right, I'm sorry.

UNIDENTIFIED FEMALE: [Inaudible].

UNIDENTIFIED MALE: Yeah. No, that's good.

UNIDENTIFIED FEMALE: Okay.

UNIDENTIFIED MALE: Yesterday, on the root server discussion, they talked about DNS root servers being able to see when a DDoS attack occurs. How would that be seen? You wouldn't see where it was coming from. How would you see that?

DAVID CONRAD: You would see it by the root server instance that you're looking at no longer being responsive. It would no longer respond to queries. If you're looking inside the machine, you would see the load on the machine just spike up through the roof. Identifying the sources of the queries is actually very challenging because one of the quirks of the DNS protocol, which I'll talk more about later, is that it uses a very lightweight underlying transport protocol that is very easy to fake, called Spoof, the source address.

So if you are using a system that allows you to muck about with the source address of a packet, you can change the source of address to the any IP address on the Internet, send the query out from your machine. When it hits the DNS server, the DNS server will respond back to that IP address that you had faked into the header, which is really good if you're wanting to attack somebody on the Internet, but it's really hard for trying to figure out where that address came from because the source address is faked.

Any other questions I can answer quickly? No.

Okay. So, how does the DNS actually work? And I have a laser. So, you're an end-user here that sort of looks like a robot working on a PC and you want to look for www.abc.com. That's sent to a machine called a Recursive Name Server. That Recursive Name Server is sort of like your agent on the Internet and it will want to first look in its internal database to see if you had asked the same question before. If not, it'll go to one of the 13 root servers. There are 13 IP addresses for the root servers. There are now I believe about a total of 500 and some odd machines that are using that same IP address across the Internet.

Your Recursive Name Server, also known as a Resolver, will pick one initially at random. As it runs longer, it'll figure out which one responds faster and tend to choose the faster responding ones to the slower ones, so it eventually learns over time to try to improve performance. It'll pick one at random, send the query, the entire www.abc.com to that server and ask, please give me, for example, the A record, the address record, of that server.

The root server will respond back that it doesn't know what www.abc.com is because the root servers only contain the top-level domains. They have no other information other than the top-level domains. So, the root server will then respond back and say, "I don't know where it is but talk to the COM servers because I know where those are. Here is the address for the COM name servers." The Recursive Name Server gets the response back, puts that information into its cache and then asks the COM servers, "Where is www.abc.com?"

The TLD name servers, if it's COM, it'll be one of the gtld-servers.net. It will respond back saying, "I do not know where www.abc.com is, but I do know where the abc.com servers are, and here's the list of those. Go ask them." The Recursive Name Server will then ask the abc.com name servers where www.abc.com is, and hopefully it will actually know the answer and return back to the recursive server, the A address for www.abc.com, which is then returned back to the requesting machine. That allows the program, for example, the browser, to then initiate the connection to the actual—well, hello. That one. You're so mean to me.

It'll actually enable the connection into that machine and you will get your web browsing experience. All of this happens very quickly. Even in a cold cache case, you're probably looking at less than 200 milliseconds, so a fifth of a second.

There was a recent study done out of the University of Southern California ISI that looked at the performance of latency of queries that go to name servers, and it turns out that the vast majority of people on the Internet are within 25 milliseconds of a root name server.

So when you're actually doing this browsing, it turns out now with very complex web pages, you're doing hundreds of queries. You don't generally notice it. It's usually fast enough that you don't see this.

Another thing that happens is that the Recursive Name Server keeps this information for some period of time. The next time the end-user asks for www.abc.com, the response back is instantaneous because they already went through all the hassle of finding out what the information was. So it doesn't need to go through that whole chain to determine what the address of www.abc.com.

Any questions on that? Yes?

UNIDENTIFIED MALE: In another presentation that we have been talking, that people can track the source IP of the query using the EDNS(0). Is this exact?

DAVID CONRAD: The source address of the query is in the layer beneath the DNS. It's actually two layers down, the IP address, the IP header, above the IP header is UDP header and then comes the DNS packet itself. EDNS(0) is an extension to the DNS that allows, among other things, for the packet size to be bigger. It doesn't contain any source address information.

One of the things that it does permit is a new extension that allows for—if you're behind like a content delivery network like Akamai or CloudFlare or any of those folks, there's an extension that allows for the intermediate system, so the resolver, that's being used to actually be encoded. That allows for the content delivery networks to make the content closer to the actual requester as opposed to closer to the resolver of the requester. But that's getting a little esoteric.

UNIDENTIFIED MALE: Thank you.

DAVID CONRAD: Any other questions, quickly? No?

Okay. So, components of the DNS. The biggest, most important part, I guess, is the authoritative server. This is what the registry typically operates. Not all the time. Increasingly, a lot of registries are hiring backend operators who actually run the DNS and databases and all that, and the registry is basically just the sales and marketing frontend. But they also own the relationship with the registry backend operators, so you can still say that the registry operates the authoritative server.

There are some historic terms that have been used with authoritative servers. The primary are also known as the master server, which is where the DNS data is actually edited. Then there's a secondary or slave server where the data is copied for redundancy and resiliency.

In this previous slide, these three servers here on the right are all authoritative servers. The root servers are authoritative, the COM servers are authoritative, and the abc.com servers are authoritative.

Go back. Go back. There we go.

The second component of the DNS is the recursive or caching server. It's a server that actually issues all the queries that go to the authoritative servers. It's what typically an ISP will run if you're using Google's public DNS. Google is providing that server.

The caching server can also be run on individual machines. For a long time, I was actually running a resolver on my laptop. It had some nice advantages. I didn't have to worry about what the ISP was going to do to the data that it got back from the authoritative servers.

Then there's another component called the Stub or the Client, and that's generally a library that's linked into the applications. So an application like your web browser or your e-mail client, anything that will accept the domain name will generally be implemented with a stub library that uses a form of the DNS protocol to send the queries to the recursive caching servers.

Talking a bit about how the DNS registry system works, if you're a registrant, you will talk to a registrar. The registrar talks via some registration interface to the registry database. The registry database collects all the changes, additions, deletions, all the modifications to the DNS database, and generates what's called the zone file where the DNS data is stored and puts that into the DNS server. The DNS server responds to the DNS queries.

The role that ICANN has in any of this, we, as the IANA function operator, interact with registries. We create top-level domains through the community-developed policy processes. We accredit registrars that are the agents, but we don't have any direct interaction with the DNS protocol itself other than the fact that we run 1 of the 13 root servers. We also try to help contribute to the development of the DNS protocols and that sort of stuff, but in general, we're not operationally involved in the operation of the DNS itself other than the fact that we run 1 of the 13.

Wrong way. There we go.

Any questions on the DNS protocol, the DNS registry, ecosystem, any of those things at this stage? Yes?

ADEL ANJI: Well, I'm [Adel Anji]. I'm a computer engineer student from Rabat. I would like to ask you if the IP version 6 will [inaudible] some changes in the DNS policy.

DAVID CONRAD: The DNS protocol was modified in 1996 to support the IPv6 protocol. In order to issue queries to get answers that had IPv6 addresses with it, we had to modify the protocol, create a new type. There are actually two new types created. One was the quad A, four A's. The other was the A6 record. A6 was an interesting idea to try to do some cute things that ultimately failed, and we fall back on the quad A, the AAAA record.

There was also a need to modify implementations, all the server implementations so that they would support queries over IPv6. By today, I think pretty much all common DNS servers support IPv6.

A number of the types in the DNS have the potential to have the potential to have address information within them as part of the data that's returned. Some of those I believe needed some modification to support IPv6, but all of that was done back in the mid-'90s once IPv6 was standardized.

Yeah. Any other questions? Okay.

STEVE: David, over that [way].

UNIDENTIFIED FEMALE: I want to ask about authoritative servers and non-authoritative servers. What are the information that an authoritative server can give and a non-authoritative server cannot? For example, if we activate the command "allow transfer", does other servers will be able to answer or only authoritative server who will be the one which answer us?

DAVID CONRAD: An authoritative server is a server that is known to be configured to accept queries for a particular domain. So if I'm running a name server, I can set up that name server to have any number of domain names on them. They're called zones. We won't go into the details of the distinction between domain names and zones, but within that DNS server, I can have conrad.com and conrad.info. There's so many words these days I've gotten confused. Cars.sucks, that's a fun one. All of those can exist on that name server, and if it receives a query for one of those names, it will answer authoritative because it's been configured for that. If the query is sent for another domain like david.com to that server, it's not able to answer authoritatively. It will say, "I don't know the answer and I don't even—," it should return a referral to .com if it knows that information, probably doesn't. It'll probably just say, "Sorry, can't help you here."

The allow transfer is a piece of the name server configuration that will—

UNIDENTIFIED FEMALE: [Inaudible] say.

DAVID CONRAD: Yeah.

UNIDENTIFIED FEMALE: Allow query.

DAVID CONRAD: Allow query. Okay, sorry. I apologize, I might have misheard you.

Allow query is actually, and again, a name server configuration that allows the name server operator to limit the clients that are going to ask it a query. That actually has nothing really to do with whether it's an authoritative or not. It's basically access control for the server. It actually is used more frequently in recursive servers because it used to be that when you've turned on a recursive server, it would answer questions for anyone on the planet. It's being friendly, being nice. If someone asks it a question, it will send a response.

That turned out to be a really bad idea because it turns out the bad guys figured that out and if you spoof the source address and send it to one of these open resolvers, then you have the ability to have a millions of machines just creating a denial of service attack.

Authoritative servers on the other hand have to be able to answer questions from pretty much any IP address on the planet because you never know which caching server is going to be asking it questions. So the allow query command actually is most usually used on the recursive server.

STEVE: Just general housekeeping, if we could. For the benefit of those online, if you could state your name prior to asking the question, that would be quite helpful. Thank you.

DAVID CONRAD: Thank you, Steve.

UNIDENTIFIED MALE: Thank you. My name is [inaudible]. I'm from Nepal. My question is how important this Tertiary DNS? Is this obviously in Primary and Secondary DNS, or we need Tertiary DNS, too? Thank you.

DAVID CONRAD: Sorry. Could you repeat your question?

UNIDENTIFIED MALE: Yeah. There are three servers: Primary DNS, Secondary DNS and Tertiary DNS, right? What is the importance of that Tertiary DNS, third level DNS? Is it necessary or it's not necessary? It should be in the same geographical area or it should be in different geographical area? You got my point?

DAVID CONRAD: Yeah, thank you

I mentioned the primary and secondary distinction was sort of historical. There have been configurations that actually had tertiary servers.

Today, it turns out that it's very rare that the primary server, which is where traditionally the data was edited is actually even queryable from the Internet. Typically in large-scale deployments these days, the primary server is hidden. It's called Hidden or Stealth Master. It's hidden behind firewalls so that the only servers that can actually talk to it are what are called the secondary servers. They're the ones that are actually replicating the data. The reason people do this is if you actually wipe out the primary master, then you have a slight problem because all the secondaries will eventually not be able to get the data that they need to copy.

So today, basically what you have are authoritative servers and no real distinction among them. In fact, it used to be long ago, when you would create a domain name, what's now Verisign, it would ask for a primary and secondary. Today, most registrars don't make a distinction. You just provide the name server.

Okay. Any other questions? No? Okay. Oh, yes.

[LISA RIMBO]: Okay. I'm [Lisa Rimbo] from Kenya and I might have had asked this question before but I didn't get much of explanation. Maybe you take me through it a bit slowly because I'm not sure a tech background.

Now, the root servers.

DAVID CONRAD: Yes, ma'am.

LISA RIMBO: If they're tampered with or if we do [away] with them, the Internet will still function, [isn't it]?

DAVID CONRAD: Sorry, if the root servers, if we turn them off?

LISA RIMBO: Yeah.

DAVID CONRAD: Okay.

LISA RIMBO: So, how does it work then? Why do we need them?

DAVID CONRAD: Okay. So, the root server system is massively overprovisioned. It has way more capacity to handle denial of service attacks and all that sort of thing than it really needs. That's necessary because denial of service attacks keep getting bigger. People have wondered what would happen if there was a successful attack. It was able to take out the root servers.

The way the DNS works—go back to this one—you have this caching name server that's sitting there. Any time you issue a query and the caching server does not know, it will go and fetch the information, pull it down and keep it for some length of time. Every caching server on the planet, once it starts up, gets the root name servers. So the information that's held in that recursive server is basically a copy of all the root server information when it starts up.

As a result, in the case of common names like com and com org, the UK, DE, most of those names will also exist in the cache. So, if you wipe out the root servers, you'll still for some period of time, be able to get your answers because the caching server doesn't need to go up and talk to the root servers. So, if you're going to try to attack the root servers, you have to maintain the attack for quite a while. And, if you're able to maintain the attack for 24 hours or a week, then you would start seeing the information that's held in the caching server time out because every record you get back has something called a time to live in it and it's only kept in the cache for that time to live. The reason for that is you want to be able to change the data eventually, so you have an ability to specify how long the cache can keep the data.

Eventually, you would start to see queries fail because it couldn't contact the root servers and you would get back a server fail, which is an error code that the DNS provides.

So in theory at least, it's not really true that you can't operate with the roots without the root servers. The root servers, they have to be there for the case where you just turned on the server and it has no other information. The first query it gets, it's going to go and start fetching that information and then it's safe for a little while.

But the reality is that the way the root server system has been deployed, the technology, something called Anycast that has been deployed, it's exceedingly unlikely that the root server system would be able to be kept down for a sufficient length of time to have a significant impact. It's not to say there wouldn't be an impact in particular. There are pockets on the Internet that would hurt badly. If you are one of the people who happened to reboot your caching server while there's a successful denial of servers attack against the root, you would probably be sort of depressed because you couldn't watch your kittens on the Internet. But in the vast majority of cases, you'd still be able to get to see the kitten websites and that sort of thing.

If there was a sufficiently interesting attack against the root, it's very likely that pretty much every Internet service provider on the planet, every law enforcement on the planet would try to stop that attack because that would be a direct threat to the Internet as a whole and people really care about the Internet these days. So it's unlikely that the attack could be maintained long enough to actually impact significant portions of the Internet.

Did that answer the question?

LISA RIMBO: Yeah.

DAVID CONRAD: Okay, good. I'm glad.

UNIDENTIFIED MALE: [Inaudible], National University of Pakistan Sciences and Technology. My question, it's extension of the question which was asked by the gentleman from Nepal. If you get the desired URL from the cache, you get it straight away. At the first stop if there is ISP, so you get your desired extension if it is there in the cache. But in other cases, it goes to the root server and if it's there and then top-level domain, and then the authoritative server. So, it means that to keep the traffic within that area, once we deployed the instances of the root server, we also need to deploy the other two servers. So it makes a set of three servers which can serve for at least restricting their traffic within that area. Is that true?

DAVID CONRAD: Yes. So, in the scenario when you want to control all of the information that flows within a particular network, you would need to be able to replicate the authoritative servers within that network in a way that would work with the applications on the Internet today.

The root is relatively easily [replicable] because it is small. Yeah, it's about 2,000 entries, I guess about 300K these days. I haven't really looked recently.

COM is, last I heard, 150 million entries. It's very large. Verisign, as opposed to the root which is generally considered public data, the registry data is considered in many cases private, owned data. I'm unaware if Verisign allows for copies of their servers.

This is something that's been identified by a number of folks in the past. The root system itself is highly resilient. It's replicated all over the place. At the top-level domain level, depending on the top-level domain, the risk actually can be fairly high. There have been instances where country codes have been taken offline because the infrastructure that they had built was insufficient to handle attacks, or in some cases, they had set up network so that a single point of failure on the network cause these name servers to go down.

In which case, if the attack had been maintained or if the network breakage had been maintained for a sufficient time, it would time out of the caches and that TLD would no longer be queryable, and now the names under it would no longer be queryable.

So, the short answer to your question is that, yes, if you want to control the information, the queries within a country, then you need a way of replicating the information, but in reality, it generally is much easier, and this is not to say it's easy. It's easier to control the resolvers, the caching servers.

SSAC actually wrote a couple of papers about this on the implications of trying to control the DNS information at various levels at the authoritative or at the resolvers. There are many hidden dangers when one tries to do this.

Do we have any water?

UNIDENTIFIED MALE: [Inaudible] one.

DAVID CONRAD: Okay, thank you.

Any other questions?

UNIDENTIFIED FEMALE: [Inaudible].

DAVID CONRAD: Nope.

UNIDENTIFIED FEMALE: [Inaudible]. Okay. Okay.

DAVID CONRAD: Oh. Oh, thank you.

UNIDENTIFIED MALE: We have the [inaudible] here.

DAVID CONRAD: Thank you very much. Was there another question? No? Okay. Actually, let me just keep going here for a second.

Now we're talking about something called DNSSEC. DNSSEC is DNS Security Enhancements.

So, what does DNSSEC do? Actually, how many people have heard of DNSSEC? Lyman, you don't need to answer. I kind of guess you know all of this stuff.

LYMAN CHAPIN: It's just a reflex.

DAVID CONRAD: It's a reflex, okay. Lyman Chapin, one of the original architects of the Internet on SSAC. Really, really good guy. Don't let him scare you.

Anyhow, it turns out that there's a little problem with the DNS. So, end-users are very rarely actually contacted through the original source of data on the Internet. They will contact the cache to get the information or there will be some firewall or something that will do the translation for them.

There's also an interesting fact about the DNS that all DNS data is transferred in the open. It's a plain text over the Net. If you happened to be in the path of the DNS query, like if you're at a coffee shop on a Wi-Fi and you're able to see the DNS queries, you can see them, you can see the query names and the types and all of that information.

And if you're a bad guy, perhaps not a bad guy but just someone very interested in playing games, you can actually respond back to the querier faster than the authoritative server and tell the querier, the stub or the client that's asking on behalf of the end-user or application, anything you want.

This is actually really helpful if you're trying to drain somebody's bank account because one of the things you could do is answer back with an IP address that puts up a page that looks very similar to a bank page and they log in, and they type in their user ID and the password. And then you say, sorry, they have no connection, but now you have their user ID and the password. Then you, on a different machine, can go and respond, go to the real bank and actually type in that information.

It turns out that if you control someone's DNS, you actually control their entry into the Internet. And if you're able to lie through the DNS, then you're able to tell them to do just about anything.

DNSSEC was originally developed to address a tiny bug in the DNS protocol that actually made that relatively easy. The DNS protocol shows a 16-bit value for something called the query ID that turns out that when the DNS was created back in the mid-

’80s, that seemed like a reasonable choice because machines were slow and networks were slow. To go through that entire space to hit a random chance that you would actually pick one of the 64 K numbers that happened to match the query remotely seemed to be very unlikely, so not a big deal.

DNSSEC tries to fix that problem because the Internet got much faster, machines got much faster and it turns out there’s a little trick that you can do that actually makes it much easier to guess those query IDs.

So, what DNSSEC does is it answers this question. There is the lasery thing. How does the end-user know the data that they have received matches what was sent from the authoritative server?

Come on. I think the batteries are dying on this or something. I don’t know. I’m pointing the wrong way is the problem. Geez, someday I’ll wake up.

A bit of history, it was originally developed in the ’90s. There were workshops with operators, Internet service providers and DNS developers and resolver operators. Through 2004, the IETF published what’s now the base documents for the DNSSEC specification in 2004.

Then there's this guy called Dan Kaminsky. In 2008, he actually did a Black Hat talk. Black Hat is a conference of people interested in finding the nooks and crannies of the Internet or, actually pretty much any technology. And, he wrote a paper called The End Of The Cache As We Know It because he had figured out basically how to corrupt any cache to insert whatever information he wanted to into the cache.

That caused people to actually become quite interested in DNSSEC. Prior to 2008, nerdy computer geeks like myself and Lyman, not to say that you're [inaudible], we would actually deploy this DNSSEC stuff, but very few people who actually wanted to do real work did. Turns out that DNSSEC actually makes a lot of things harder. Security in general makes everything harder. DNSSEC took that to the next level.

With a resolver, generally, all you had to do before DNSSEC was turn it on and it magically did everything you needed it to do. You didn't have to do much configuration. When you turn on DNSSEC, you actually have to configure a piece of information called the Trust Anchor. That trust anchor, you have to make sure it's up-to-date. Fortunately, it doesn't change that frequently.

On the authoritative side, turns out that signing zone data to actually generate the cryptographic signatures turns out to be really hard and in numerable times, top-level domains and second-level domains have gone off the Net because people didn’t do signing correctly.

People who actually want to do real work as opposed to play around stayed away from DNSSEC until 2008 when Dan demonstrated that was really easy to poison caches, to insert whatever information you want to do into a cache. That caused people to become much more interested.

Since 2009, DNSSEC has been turned on in most top-level domains. The root itself was actually signed in 2010. Today, I believe about 70% of the TLDs are signed. Of course, with the root being signed, that allowed for what’s called a chain of trust to be built. Today, I believe about 3% of the second-level domains are signed, which is not a big number, but it is growing.

Let’s see.

How does DNSSEC actually work? Well, before I talk about having these, what are called DNS records, the query name, the type, the information that you receive back, the DNS response data is accompanied with a digital signature. That digital signature can be validated with a public key. That public key right now is the one that’s held at IANA.org. I can provide the URL for any of you who are desperately curious.

The key that allows the resolver to ensure that the data that is being received has not been modified in flight. Authoritative servers respond to queries with the signed answers, so it returns the answer and it returns the digital signature associated with that answer. So when the resolver receives that information, it gets the answer then it computes the same signature and sees if the signature matches what was sent in the packet. If it doesn't, then you know somebody mucked with the data in the response, and that actually allows you to build a scalable trust network using the DNS.

If your registry, what do you need to do with DNSSEC? Well, your registrants, all the folks who've bought domain names and went to the trouble of signing them, they actually have to provide information back up to the registry so that gets put into the registries information. These are called Delegation Signer Records. This turns out to be something of a problem, which I'll explain later.

The registry has to sign and publish those DS Records into their zone. They have to sign negative answers. So, with the DNS, it turns out today especially, getting a positive answer is actually relatively rare. There are a lot of bad pieces of software out there that are generating garbage names that they're thrown to the DNS and almost all of those are return does not exist, the NXDOMAIN response.

And, the reason for that is, these are pieces of malware that are attempting to contact a command and control server. They make up a dummy name using something called a domain generation algorithm and send the query out. One out of a thousand might actually work, and that sets up a connection that allows the malware to turn your computer into a zombie and be part of what's called Botnet.

So, there are a lot of these NXDOMAINS and DNSSEC allows the zone owner to be able to say, "This name does not exist and you can prove it." The registry also manages the keys for the top-level domain, and for a top-level domain, they interact with IANA to register those TLDs to create this ability for their clients to submit DNSSEC records to create a chain of trust all the way up to the root that then allows the resolvers to validate going back down.

Wow. I've been talking too much.

The registrar submits the registrant's DNSSEC information to the registry. When a registrant goes and signs their domain name, they'll get a DS Record. They'll have to provide that up to the registrar, either as the Delegation Signer Record or the actual key itself. The specification allows both.

This actually presents a little bit of a problem. In the original design of the DNSSEC, there was not this registrar/registry split.

You would talk to the registry directly, so that meant that it was the registrant talking to the registry and there wasn't this third party, this trusted intermediary.

But what happens if the intermediary isn't trusted? With DNSSEC, you actually have to trust the registrar to provide that information and not corrupt that information, not make a boo-boo and put in the wrong information. In general, that was okay because you have to trust your registrar anyhow because they manage the name server records. But it turns out today, there are a lot of backend registrar service providers that want the ability to update the names on behalf of their clients.

The problem is that every registrar does this differently, so if you're one of these companies, you have to talk to each of the registrars and figure out how they accept information. That's something that right now is being discussed in some places within the ITF and ICANN to figure out if we can come up with a solution.

Stop it, come back. Wrong way. Wow. Someday I will wake up and then I'll be scary.

Looking at sort of the interactions here, as mentioned, you have the registrar that's the registrant's agent. They do various DNSSEC signing functions. If you're a TLD that talks with the root, these go into the registry's database, goes into the DNS server and this enables the DNS queries to actually be DNSSEC protected. That was the DNS part.

Any questions, comments, screams of outrage? Lyman? No? Good. Then I will move right along because time is growing short.

The next protocol is actually sort of an embarrassment. This is WHOIS. Anyone here actually deal with WHOIS? Okay.

WHOIS is very old. It actually predates the Internet. It was originally developed to allow the network researchers on the ARPANET to know who to contact when bad things happened to the network. It is an unbelievably simplistic question and answer protocol, and it was defined at a time when we didn't bother with minor details like privacy, security, or even accuracy because we were all together. Network researchers, we pretty much knew each other and we'd go to interesting places and have meetings, and yell at each other about three times a year.

Wow. Here we go.

What is this incredibly simplistic protocol? Well, this is it. You open a TCP connection – TCP is the sort of a virtual connection – to port 43. You send the question, you wait a little while, you receive an answer and then you close the connection. In many cases, after you receive the answer, you actually display it to the user, but that actually isn't a part of the protocol. That's it. That is the some total of the WHOIS protocol.

Now, why is that a problem? Well, this is intended to show the relationships, but it's WHOIS and it's painfully simplistic. So, the problem is that what's in a question and what's in an answer is undefined. It's not specified in the protocol, and that means that it's free form and it allows registries and registrars to do anything they want, and they have. They've done anything they've wanted. This means that WHOIS servers and WHOIS clients don't interoperate very well.

Also, the WHOIS protocol uses something called Netascii, which is a simplified form of ASCII and does not even come close to understanding anything about internationalization. Through the protocol, there's no way of interoperably handling any Arabic script or Chinese or anything like that. There's no way in the protocol to provide sort of a [matter] answer like the way the DNS does, "I don't know the answer but check over there." WHOIS doesn't support that. There's no way to provide user credentials. This means that there's no way within the protocol to say, "Well, I only want to give you part of the information because of who you are."

This is a problem in Europe. If you're in a place with a data privacy statute that says you can expose personally identifying information, and you consider their home address personally identifying information, WHOIS doesn't give you any way of distinguishing whether someone issuing a query is authorized to get the personally identifying information or they should be sent the minimal information that's legal under the data privacy statutes. This is something that we have to deal with in ICANN's context because we force the registries and registrars to use WHOIS and strangely enough, there are registries and registrars that aren't in the US who has very weak privacy laws, and there are registries and registrars in countries that have very strong privacy laws.

So, what happens? Anyone who actually deals with WHOIS in an operational sense ends up hating it with a passion because it makes life very hard. There are a number of WHOIS servers who generate different forms of output, which means you can't easily programmatically parse it. You can have a standard user interface across all the TLDs and registrars. This lack of differentiated access makes all sorts of really fun and interesting legal problems that ensures that lawyers always win.

So, what happens? The IETF, the folks who standardize all these protocols decide, "Well, we'll fix that. We'll come up with a new protocol." They tried four times now. The first one was something called WHOIS++, which was sort of a geeky joke off WHOIS. It's an increment of WHOIS. That didn't work because people would have to deploy new software and new infrastructure, they didn't want to do that.

Then there was something called RWHOIS, which was Referral WHOIS, which dealt with the metadata. If I don't know the answer, go look some place else. RWHOIS dealt with that but then didn't deal with the other questions, other challenges. As a result, it actually got some deployment. One of the regional Internet registries, the folks who hand out IP addresses actually allow their customers to use RWHOIS to minimize their interactions with their regional Internet registry. So it get some traction but eventually failed as well.

There was something called IRIS, which was yet another protocol that try to build a different infrastructure. It failed horribly.

And then, the regional Internet registries, two of them got together and created something called RDAP, which stands for the Registration Data Access Protocol. It is a query response protocol that's used to inspect a registration database. What the guys at the [inaudible] figured out is if you layer the stuff on top of the web, because the web is everywhere, then you already have this huge infrastructure that's already built and you can just get beyond the questions of, "How do I beat the chicken and egg problem of deploying infrastructure?" You also reuse a lot of the existing technology that's been deployed for web-based transactions.

The components of RDAP, obviously, you have software to receive and parse queries, something to access the database and send back responses. And the client, well, the client simply is something with a web browser API like a web browser. You can query a RDAP server just by pointing a web browser and putting in a particular string.

History, as I described, people are unhappy with WHOIS so the couple [inaudible] folks came up with this new thing called RDAP. Then they went to the IETF, which turns out to be a good way of doing things. You come up with something that sort of works and then you take it to the IETF, and they standardize it and make sure it's interoperable.

As a result of all of this, RDAP is actually very tightly tailored to be a replacement for WHOIS that provides a commonality across names and numbers. RDAP has a mechanism to query both domain names and IP addresses, as well as autonomous system numbers. And it's also secure because it uses HTTPS.

So basic description, it looks like a URL. That first one there is how you look up an IP address. Whatever the server is, you just put it there and using this format for the string, that's how you look up an IP address. This is one way of looking up a domain name. The response comes back over HTTPS and looks like this. Isn't that intelligible?

If you take a web browser, here we are, and put that into the string, that is the answer you will get. It's actually four pages of that. And, for a computer guy, that looks pretty cool because that's actually parsable and you can do stuff with it and it's not freeform text. But there are RDAP clients that actually make that look prettier and do other things.

The features of RDAP, as I mentioned, one of the big problems with WHOIS was anyone could send back anything they wanted and it was still conformed to the protocol. RDAP fix that. They actually define the data model. The data model is actually quite extensible and it supports the entire UTF-8 repertoire. It allows for distribution of data sources. Since it's based on top of the web, you have referral objects that came with the protocol itself. It allows differentiated access because it's on top of the web. You can have authentication using the web. And it's the way we do software development these days.

As I mentioned, these aren't really my slides and I'm sure Ed would actually talk about this but I'd just skip it.

That's RDAP. Does anyone have any questions?

STEVE: Or you can use the desk mic.

UNIDENTIFIED MALE: [Inaudible] National University of Sciences and Technology, Pakistan. The purpose of WHOIS was to facilitate the administration like the domains, the [inaudible] those have been registered. So, don't you think that once you have a recursive process of identifying the requirement and then getting a protocol on or getting it on commissioning the protocol for in the system, one finds there is a gap because not the data which it provides, if everyone can access that, probably that was the main concern in case of WHOIS. Anyone can just make a [query] and can get the data that this site is [inaudible] at this place.

So, how you relate it to the initial requirement like RDAP versus that problem if it implements [AAAA], right? Knowing now these are the authorized people and there is a procedure to allow these people. With that first, what was the intent? Because it doesn't has that relationship with the operational activity. It's just a sort of administrative tool to facilitate certain procedures.

DAVID CONRAD: Right. As I mentioned, WHOIS was originally developed back in the ARPANET days to allow network researchers to find each other. If they were working on a protocol and some remote IP address was doing silly things, they could look up that IP address, find out the owner of the machine to which that IP address was configured, call them up and say, "Hey, your machine is doing stupid things," and they'd work it out and figure out what the problem was.

You are quite correct that WHOIS, since it has no authentication controls, allows you to basically scavenge the entire database and get all of the contacts associated with every object within that WHOIS database. That is a problem and that's one of the reasons people came up with RDAP is that they wanted to allow for a differentiated access so that if you didn't have credentials, you could be rejected. You could say, "I'm not going to give you that information because you did not provide the appropriate credentials for that level of information."

UNIDENTIFIED MALE: Is it true that there are still some registrars and registries who are maintaining this database of WHOIS or it's—

DAVID CONRAD: Well, so, that gets into the Thick versus Thin discussion, so the Thick WHOIS versus Thin WHOIS.

Traditionally, there was one registry. In the dim, dark history of the Internet, there was one registry. It was called the NIC and it held the entire database. When we went to a split between the registry and the registrar, the data that the NIC held needed to be broken up. We had multiple registries, so each registry had its own WHOIS database. And then, when we added registrars, there was a decision that had to be made whether the registry held the data or the registrar held the data. Because the registry doesn't talk to the registrant, right? There is a layer there.

If the registry holds the data, then there may be a problem in keeping that data up-to-date. If it's a registrar, then there's a direct relationship and the registrar said, "Look, you can call up the registrant and say, 'Look, you need to update the data.'"

This is still a question within the ICANN world. There are three thick registries: com, net and one other I thought, org—okay. So yeah, so there are three thick registries: com, net and org. Everyone else is thin. What a thin registry is, is—sorry, I have that reversed. Apologies. Com, net and org are thin. The only information they have is the domain name and then the registrar where to go to actually get the WHOIS information. The registrars hold all the registrant information. In the thick model, it's the registry that holds all the information.

Currently, there is a policy that is attempting to move all the registries over to thick, but that actually is a problem because it runs under data protection laws.

I don't know if I answered your question but that's that situation.

Any other questions, comments? Screams of outrage?

Very quickly, talking about the extensible provisioning protocol, this is a business-to-business protocol that's used between a registry and registrar. This is how registrars edit the registration database. Traditionally, registry owners would go in and edit using Emax or some text editor and then fire up their name server, and it would suck in the data. When we went to this registry-registrar split, we needed a protocol to allow the registrars to edit the registry's database remotely. That's what EPP is.

Its history originally developed in the IETF, then standardized in 2009. It is mandated by ICANN for all gTLDs and sTLDs. It actually derived out of an earlier protocol developed by network solutions, which became Verisign called RRP, Registry Registrar Protocol. It is gaining acceptance within ccTLDs primarily because ccTLDs want to sell their names and it's easier for them to sell their names if they use the same provisioning protocol as the gTLDs, so increasingly, ccTLDs are using the EPP.

And there's actually a working group within the IETF that is managing the extensions. EPP, Extensible Provisioning Protocol, means that people can make changes as much as they want. Let's see.

What else to say? Go back. Here we go.

EPP uses TLS. TLS is a way of creating a secure connection and it's strongly secured. The communication that EPP, the data that it sends back to some is sent using the Extensible Markup Language, so everything about EPP is extensible. Obviously, the server for EPP is inside. The registry, the clients are at the registrars.

How much time do we have here?

STEVE: Two minutes.

DAVID CONRAD: Yeah, so I'm not going to go into this.

STEVE: What I would suggest is that we had an archive of the presentation from yesterday. So unfortunately—well, fortunately, there were some great dialog today. Let me start with the positive.

DAVID CONRAD: Very much appreciated the questions.

STEVE: A really good dialog today. Unfortunately, that great dialog cut our time. And, the archive from this session yesterday will be on the meetings.icann.org agenda and you can drill into it and see what we said about data escrow and Trademark Clearinghouse.

DAVID CONRAD: Right.

STEVE: I think those are only the two left, right?

DAVID CONRAD: Yup, data escrow and Trademark Clearinghouse. Actually, this is probably for the best because I'm really weak on this. I never bothered with this stuff and I was just sort of rambling mostly yesterday.

So, I want to thank you all very much for taking the time and participating, especially all the questions I very much appreciated. I hope I answered them in ways that didn't make it worst but I'm happy to answer any questions you might have.

Feel free to send me e-mail. I think the last slide have my e-mail address.

STEVE: The last slide.

DAVID CONRAD: But it's david.conrad@icann.org. I'm happy to answer any questions anyone might have.

And again, thank you very much for your attention.

STEVE: We're going to take about a 15-minute break here and the next session will be on the Root Server Operations. If you guys have an interest in Root Server Ops or the Root Server System Advisory Committee. Sorry, it's been a long day for me, too. RSAC.

It will be 15, so go fast. We're tight on time right now. So please, stay or come back or whatever.

[END OF TRANSCRIPTION]